Whitepaper

Mobile Test Automation

Solutions to the Mobile Test Efficiency Challenge

sqs.com

Authors: Sven Euteneuer
Senior Research Manager
SQS Software Quality Systems AG
Germany

Abhijeet Padwal
Head of Talent Management
Associate Director
SQS India Infosystems Pvt. Ltd.

With contributions from
Himanshu Chandra, Parag Kulkarni
and Nandini Sarma

Published: August 2013

SQS – the world's leading specialist in software quality

SVEN EUTENEUER

Senior Research Manager
sven.euteneuer@sqs.com

Sven Euteneuer graduated from the University of Bonn, majoring in Computer Science, and has been active in IT projects in various roles, with a focus on quality assurance, for over 18 years. After joining SQS as a Senior Consultant in 2007 he has delivered projects in the domains of architecture analysis, code analysis and security testing for a wide range of customers. He is currently responsible for the services in the Mobile Testing space.

ABHIJEET PADWAL

Head of Talent Management
Associate Director
abhijeet.padwal@sqs.com

Abhijeet Padwal studied Computer Science and has been with SQS since 2000. He is Head of Talent Management Group and Performance testing CoE in SQS India. His core competencies are in the area of non-functional testing and predominantly in load performance testing. He has also been instrumental in exploring and defining new service offerings for SQS.

With contributions from:

HIMANSHU CHANDRA

Senior Test Analyst
himanshu.chandra@sqs.com

Himanshu Chandra holds a Bachelor degree in Engineering in the field of Computer Science and has been with SQS since 2011. His core competencies include Automation Framework design, Test Suite Management and Automated Test Case creation. He has worked extensively in the Telecom domain over the past six years. Of late, he has been actively involved in the Research & Development of automation of Mobile applications which are based on the Android platform.

PARAG KULKARNI

Test Lead
parag.kulkarni@sqs.com

Parag Kulkarni holds a Bachelor degree of Computer Science and has been with SQS since 2008. His area of expertise is in the field of Manual as well as in Automation using a variety of tools. His core competencies include Test Management, Automation Framework design, Automated Test Suite Management and Automated Test Case creation. He has worked extensively in the CRM, Publication and Telecom domain over the past eight years. He voluntarily got involved in R&D for Android Mobile Apps.

NANDINI SARMA

Senior Test Analyst
nandini.sarma@sqs.com

Nandini Sarma is a Computer Engineer and has worked with SQS India since 2009. She has over five years of experience in Virtualisation and Retail-Supply Chain Domains. Her areas of competency include Automation Framework designing and management, Automated Test script development and maintenance and she has been involved with Code Quality Testing in the recent past as well. She has been involved with R&D of Automating Mobile based applications on Android platforms.

Contents

1. Management Summary

This whitepaper focuses on two case studies, illustrating successful test automation solutions for apps on mobile devices such as smartphones or tablets. In this still emerging and dynamic market, phones running a number of different operating systems or platforms are available. Subsequently, mobile app providers are often unclear on how to deal with issues such as test automation in such a volatile setting.

On the other hand, many established test automation solutions do not yet offer the desired level of support for mobile apps and often involve a too complex boot-strapping before they become effective. Open Source tools require deep technical knowledge and are usually only available for a single mobile platform.

The two solutions presented in this whitepaper – NeoDroid and SQS TC – were both created to answer specific customer or project requirements – requirements that no available automation tool could provide at this time. Both tools are based on the same core automation framework that allows them to provide test automation for native apps on the Android and Apple iOS platforms. Apart from this shared core, the two tools offer quite different solutions, however. While NeoDroid is aimed at providing a modular, scalable and centrally manageable test automation solution for large-scale, long-running app development effort, SQS TC is aimed more towards the typical mobile app: small, nimble and focused on implementing a single capability. Here, qualities such as time-to-market, quality and efficiency become more important than reusability, for instance.

The two tools demonstrate that effective and efficient mobile test automation solutions can be put in place already, if context and requirements are well understood and if the best fit is sought and implemented.

This whitepaper covers the confluence of two success stories in IT: Mobile Devices and Test Automation. In order to set the stage for an in-depth discussion about the fascinating aspects of applying test automation to the mobile space, we will first revisit core concepts of both mobile and automation.

1.1. Mobile Devices and Testing

Few trends have changed the IT world so swiftly and yet so profoundly as the move towards mobile devices which began in the first decade of this century and which has gained enormous traction over the course of the last three years. Yet, mobile phones and even the combination of phones and organisers, dubbed smartphones, have been around for a while: Modern digital mobile phones penetrated the world's markets in the 1990s and companies such as Microsoft and Palm have offered versatile phones with a large feature set for quite a while.

However, the real hype started with the introduction of Apple's iPhone in 2007 and the competing platforms from Google and – to some degree – the renewed attempts by established players such as Microsoft or RIM.

These platform manufacturers are still battling for market share, with the lion's share of the market being split between Android-based phones and the iPhone. In a second wave of innovation, the concepts that worked for smartphones were applied to tablets, devices with a much larger display and less focus on telephony as a use case.

Since then, these mobile devices have had a profound impact on IT. Among the most visible changes was the adoption of a different mindset concerning the use of software; the concept of having many small, specialised apps instead of large, monolithic applications as well as the idea of centralising sales and distribution of these apps through a single source – the app store – has resulted in a wide array of impacts (cf. Jacob & Tharakan, 2012).

Testing is impacted by these changes as well, as more and more IT organisations are asked to either provide apps for mobile devices or are in a situation where they need to support the use of mobile devices as part of the organisation's business processes. Subsequently, few IT organisations will be able to escape the question of how to best test mobile devices.

In the end, mobile testing can make use of the large body of knowledge that exists for testing software in general, as nothing inherent to mobile devices invalidates testing techniques and methods defined in the past. However, mobile testing differs from the test approaches established in many IT organisations in a few key areas:

1. **Mobile Systems Requirements**
   Compared to stationary computers, mobile systems bring with them a range of new requirements that companies need to take into account. Smartphones, for example, are generally not connected with the rest of the company via a secure network, but use GSM, UMTS or a public WLAN, depending on the location. In addition, there is often a range of different, some privately purchased, devices. Companies need to take these into account as part of their requirements analysis.

Business and quality assurance experts also need to work together to define requirements as this ensures that companies systematically and verifiably determine what properties any mobile software should have. In addition, the correct definition of requirements accelerates software development and testing.

2. **"Quality Fingerprint"**
   In principle, the quality criteria of mobile and traditional software are no different. The focus, however, shifts to functionality, security, performance, ease of use, reliability, portability and maintainability. These criteria need to be weighted differently for mobile solutions creating a different "Quality Fingerprint". Mobile software, for example, has to be as sparing as possible on hardware resources, to extend the battery life. The subject of security also needs to be taken even more seriously with mobile systems than with traditional IT, as mobile devices come with substantially more interfaces. In terms of hardware, this may be WiFi; in terms of software, for instance, social media apps. Mobile systems permanently exchange data over the internet via these interfaces and each additional interface increases the security risk and is a potential gateway into the device, and thus into the company's IT.

3. **Multi-Channel Testing**
   The basic principles of software quality assurance and software testing remain the same with mobile systems. Companies should therefore not reinvent the wheel and, under no circumstances, introduce separate quality assurance (QA) for mobile systems. What's more important is to make the existing QA more systematic, to ensure that the particularly short time-to-market can be achieved with mobile systems. In particular, the QA must ensure effective management of the many types of mobile software, e.g. different operating systems like iOS or Android, but also different device classes such as smartphones or tablet PCs (cf. VisionMobile Ltd., 2012).

4. **Testware**
   The traditional procedures of software development and QA continue to apply to mobile systems. However with iOS, Android and the like, traditional tools can often no longer be used. Additionally, new testing tools need to consider being able to test quality criteria such as security or efficiency, which were, until recently, not taken into account. One example of this is "fuzzing", which bombards the interfaces of mobile systems in a kind of stress test, with the aim of breaching them. In this way, "fuzzing" can detect any problems in the areas of security or robustness.

5. **Development and Testing Processes**
   When developing and introducing mobile systems, flexible process models are essential, as mobile products change more quickly and frequently than traditional information technology (cf. Wikipedia, 2013). Iterative and incremental developmental models are recommended, as they test the system requirements and their implementation much more frequently than sequential procedures. In the selected process model, the user should also be closely involved in the definition of requirements. Combined with early definition of requirements and a systematic approach to quality assurance overall, this substantially reduces the time-to-market.

1.2. Mobile Test Automation

Due to this increased frequency of changes, mobile development teams need to look into ways to increase the efficiency of the development process. Tool support and automation of repetitive tasks can leverage efficiency gains and become attractive

options due to this. Testing – a resource-intensive activity – is often targeted by such attempts. Test Automation is understood to comprise of (ISTQB, December 2007):

> The use of software to perform or support test activities, e.g. test management, test design, test execution and results checking.

Definition 1: Test Automation

Usually the strongest focus is on the automation of test execution. For traditional desktop or server software this often means the automation of command-line or GUI elements, with GUI automation being more complex due to the need to correctly identify and execute the necessary activities on the GUI to execute the test cases. Subsequently, a variety of approaches on how to implement such a GUI automation have sprung up, the most prominent being capture & replay, data driven and keyword driven automation.

Among these, capture & replay is the most basic technique consisting of a capture phase where the test automation tool supervises and analyses the activities of manual test case execution and a subsequent replay phase, where the captured information is used to re-enact the test case execution without user intervention. Data driven automation, on the other hand, uses pre-defined data sets to drive the test automation execution. Lastly, keyword driven automation utilises reusable placeholders so that any given UI element can be used in a variety of test cases and with different sets of test data.

Each of these approaches has its merits with capture & replay as well as data driven automation providing quick results with limited need for specialist knowledge. Keyword driven automation, on the other hand, is much easier to maintain in the long run.

For mobile devices, these approaches work similarly, albeit with additional aspects that require consideration. For instance, mobile apps can be tested on a physical device, but more often than not other options are available as well, ranging from virtualised devices in a test lab to device emulators running on PC hardware. Additionally, the mobile market is not yet as mature as the desktop PC market, meaning that there is a wider range of competing platforms out there, resulting in the need for tools to either focus on specific platforms or to cover a diverse array of different technology stacks.

2. Market – Current Status and Outlook

Mobile continues to drive a large share of the IT market. According to market analyst Gartner, sales of mobile devices have been stable above 400 million units/quarter for 2011 and 2012 with small variations accounted for by fluctuations in demand, update cycles of key models such as the Apple iPhone and changes in the world economy. Gartner Research estimates the total number of mobile phones in use worldwide to be ~5.6 billion with China alone accounting for 950 million mobile phones in use. In the third quarter of 2011 one in every four mobile phones sold worldwide was a smartphone, with the ratio of smartphones to feature phones projected to increase further (cf. Gartner Inc.)

This renders mobile devices among the most potent drivers of IT spending, outpacing more traditional sectors such as PCs. At the same time, a number of industries (such as banking, retail and travel) are looking into extending self-service options or are looking to foster customer relationships by offering (often free) apps for mobile devices that customers can download and install. Often these apps increase efficiency or convenience of everyday tasks; for instance by making use of specific features of mobile devices such as location-awareness. Other industries, in turn, may want to distribute the same pieces of content not only to websites, stationary or physical media but also to mobile users. Video clips, podcasts, music or digital newspaper editions are examples of this. Revenues from mobile apps alone are estimated to approach USD 30 billion by 2013 by Gartner Research, including both paid-for apps as well as revenue from in-app ads.

While this trend may have initiated among small, tech-savvy start-up companies and then spread to the media industry, it is now reaching much broader, more traditional and conservative sectors of the market such as banks, insurance companies or utilities. Many of these organisations do not yet have a clear picture of how to deal with this new phenomenon and turn to SQS in their quest for a mobile strategy. In Figure 1 a typical overview of this new type of IT is given for the financial industry. 52% of the mobile users use their phones to check their balance, 42% use it for financial transfers etc. And these numbers stem from the end of 2010!

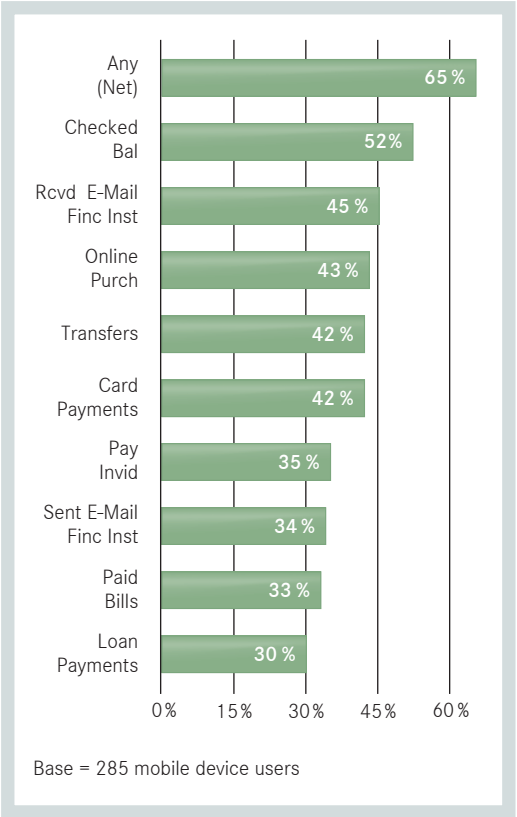


Figure 1: Top financial activities conducted on mobile devices

When looking at the internal structure of the mobile market, the first striking aspect is that it still consists of a fairly large number of competitors, unlike the desktop PC market where only two contending platforms have survived. Table 1 gives an overview of overall sales per smartphone platform as well as the growth compared to the previous year.

Operating System	Sales 2011 (in € m)	Annual Growth
Android	237.7	244%
iOS	93.1	96%
Symbian	80.1	-29.1%
Blackberry	51.4	5.0%
Windows Phone	6.8	-43.3%
Market Total	**469.1**	**54.5%**

Table 1: Smartphone Sales by Platform
(Source: Canalys, 2012)

Android can be seen to gain a lot of traction and has already captured the majority of new device sales. Apple's iOS-based smartphones are the distant second and are followed by the remaining platforms.

Table 2 shows the situation for the tablet market, where Apple is still enjoying a lead over Android which is catching up quickly.

Operating System	2011	2012	2013*	2016*
iOS	39.99	72.99	99.55	169.65
Android	17.29	37.88	61.68	137.66
Microsoft	0	4.86	14.55	43.65
QNX	0.81	2.64	6.04	17.84
Others	1.92	0.51	0.64	0.46
Market Total	**60.01**	**118.88**	**182.46**	**369.26**

Table 2: Tablet Sales by Operating System
*estimated sales (Source: androidguys, 2012)

All in all, it can be said that the mobile market as a whole is growing with a high pace. Worldwide growth will continue for the foreseeable future, fuelled by demand from emerging nations, from tablet market growth. Within the market, Android is clearly the platform that will dominate the next two years. It is not yet clear, however, who will dominate the market in the mid-term. Android, iOS and Windows Phone all have the opportunity to become the market leader in this highly volatile market.

3. The Challenge

Test automation on mobile platforms is still a very young market. A variety of tools are already marketed but these suffer from a range of issues that prevent their use on specific platforms or reduce their efficiency. The following sections will discuss the approaches, their issues and potential solutions.

For producers of mobile apps, however, this means that more often than not they will need to consider releasing their products for more than a single mobile platform, thus increasing development efforts.

In order to limit the effort necessary to test different variations of the same app for multiple platforms, a comprehensive test strategy involving the targeted use of test automation for key platforms is required. In the preceding months, a number of mobile automation solutions have appeared on the market to answer this demand. Usually, test automation will break even after the fourth test iteration (cf. Figure 3).

Additionally, most mobile apps differ from full-blown PC applications in that they are very specialised tools that aim to do a single thing well instead of many things at the same time. For instance, a typical app provides a weather forecast but does not provide features to write emails or jot down notes at the same time – other apps cater for these use cases.

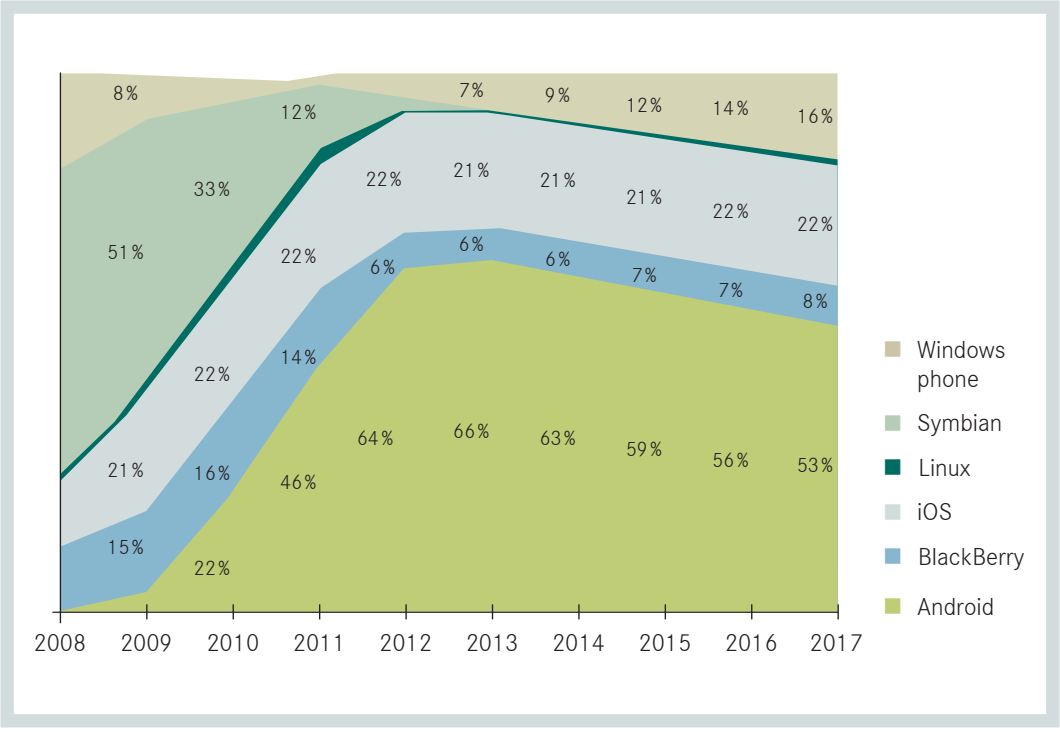


Figure 2: Mobile platforms market share (Source: Geronimo, 2013)

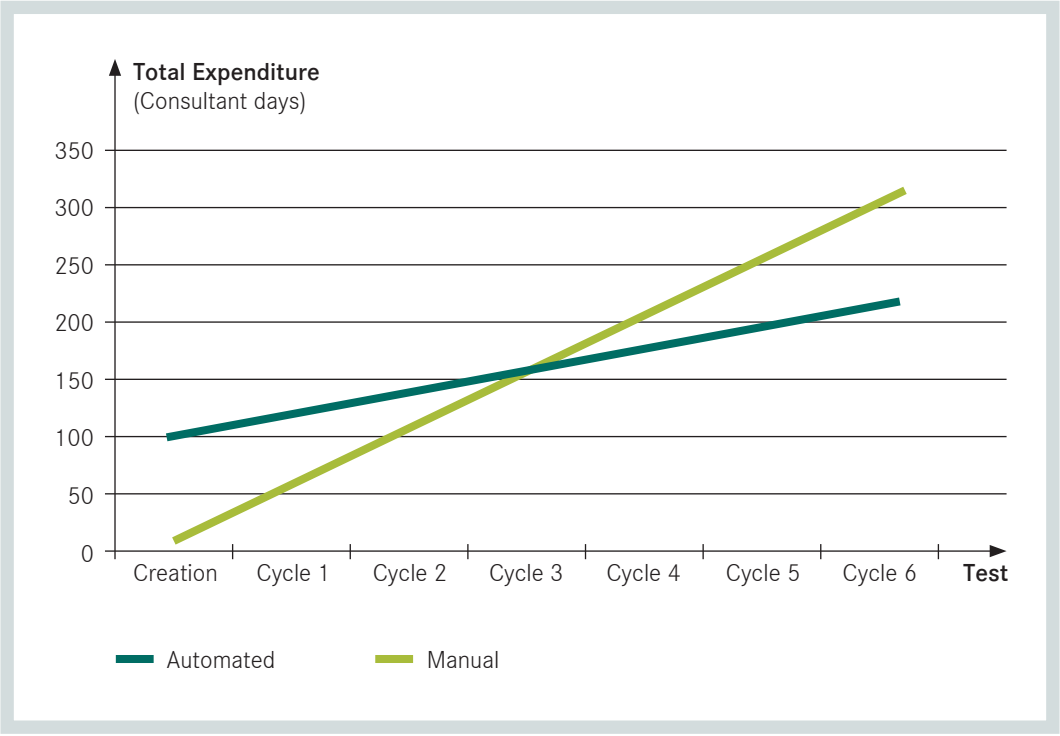


Figure 3: Break even for test automation

As a consequence, apps tend to be small-scale pieces of software, often developed by small, agile development teams. This approach requires similarly agile tools that support small-scale apps, short iterations and quick time-to-market. Sophisticated and complex tools that have a merit in more complex, larger IT projects fail to add value for such users.

3.1. Issues with Traditional Automation Approaches

The majority of established commercial test automation solutions for mobile devices are based on three core design decisions:

1. **They record and generate scripts with hard coded values.**
   This implies that it becomes difficult to maintain and evolve automated test suites. This is due to the fact that such an approach will result in automated test cases that do not clearly separate between automation infrastructure that is used by many test cases, test automation objects such as GUI elements and test data. If any one of those elements changes, all affected test cases need to be changed, decreasing maintainability and increasing the required effort just to keep the test suite working. In the long run, this will lead to a situation where there is no return on investment for automated testing anymore, due to the costs of maintaining the test suite dwarfing the cost savings.

2. **They generate binary packages, require the application under test to be available in specific formats and often require rooted or jailbroken devices.**
   The implementations of the existing solutions dictate that they can only be used in specific technological contexts. This often includes the need to circumvent built-in security mechanisms – called rooting in the case of Android devices or jailbreaking for iPhone and iPad. Additionally, existing implementation often requires the availability of specific artefacts such as the installation package of the application under test. This, however, may be difficult or even impossible to provide for third party apps that are not under the control of the testing organisation.

3. **They execute those packages and generate results based on bitmap-comparison of screenshots.**
   The actual execution consists of running the prepackaged test cases, i.e. replaying the recorded actions on the mobile device. In order to establish whether the test case execution yielded the expected results, the current generation of tools relies on the comparison of screenshots with predefined images. This approach is not only less flexible when it comes to reacting to changes in the GUI of the application under test, but it is also less precise as in some cases the bitmap comparison of screenshots cannot correctly evaluate the results of a test case execution.

In addition to the challenges posed by these design decisions, additional aspects render this crop of tools difficult to implement and use. The most significant of these challenges is probably cost – most of the existing solutions are add-ins to more comprehensive test management tools, which, in turn need to be paid for as well. This way, the prospect would need to pay twice – once for the tool and a second time for the test automation add-in.

Free and open source test automation tools may seem like an attractive alternative from this point of view. However, even though there is usually no need to pay license fees for usage of these tools, the reduced cost comes with the price of reduced functionality: Open source solutions such as Robotium for Android are not comprehensive enough to cover all relevant requirements for mobile test automation. For instance, Robotium only covers a small subset of user interface elements (such as buttons, lists, input elements, etc). Applications that make use of the full set of UI elements cannot be automated out of the box. This bare-bones approach surfaces in a multitude of areas, requiring the adoption of custom-built frameworks around the open source solution that ensures that all required functionality is there.

Last but not least, virtually all existing mobile test automation tools require extensive technical knowledge in order to use them properly. However, this knowledge is often available only with specialists which are a scarce resource in most organisations. With mobile apps however, this shortage of automation specialists leads to long lead-ups being necessary in order to plan the availability of key staff or delays because staff members have been reassigned to more critical projects, leading to sluggish project performance and an unnecessarily long time until apps can be released.

3.2. Potential Solutions

Subsequently, the ideal mobile test automation solution should tackle all of the abovementioned issues and challenges:

- It should be low-cost, avoiding high license fees as well as high operational expenditures such as maintenance and support fees or internal effort necessary to maintain the solution.

- It should be modular and easily upgradable so that migrating to new trends in the mobile market is not hindered or prevented by lack of support from the test automation platform.
- A mixture of technical and non-technical team members should be able to create, maintain and execute scripts. It has been shown that ideally, a test automation team consists of technical as well as non-technical roles so that both good technical and subject-matter knowledge are available within the test team. Having a test automation solution that is accessible to non-technical team members greatly eases accomplishing this.
- It should support creation and maintenance of individual test cases, but also the creation, maintenance and management of whole test suites.
- It should support the management of test dependencies and modularity, e.g. being able to define interfaces to user interface elements only once so that changes to the application can be reflected by changing only this interface in the test automation suite.
- It should support the management of test data separately from the automation scripts so that test data sets can be changed with little overhead.
- It should support test case execution on multiple mobile device platforms so that a single automation environment can be used to test all relevant devices.
- Last but not least it should support the execution of test suites in batches and should provide facilities for customisable test reporting.

In the following sections we will introduce two case studies. These case studies illustrate how solutions providing the above capabilities can look. They also illustrate that there is no need to make use of the big name tools in automation to build reliable, effective and efficient automation frameworks.

The tools in both case studies are based on a core automation framework that can be used as part of an overall solution to a specific problem. In this sense, both case studies illustrate different use cases:

- NeoDroid was created to provide reusable and centrally managed automated test cases for large-scale mobile software development projects, i.e. feature-rich apps where development would go on for long stretches of time.
- SQS TC, on the other hand, was created to provide light-weight test automation facilities for small focused apps, where platform-coverage, time-to-market and quality were paramount success factors and complex automation frameworks would not have delivered value.

Both case studies discuss the approach in more detail, give insight into the respective benefits and show where both tools still have potential for future improvement.

4. Case Study: NeoDroid

While the abovementioned list sounds somewhat like an unrealistic pipe dream, the vast majority of it can in fact be fulfilled today by NeoDroid – the solution discussed in this section – the only requirement remaining unfulfilled being the ability to execute test suites on multiple mobile platforms. However, even though the current version of NeoDroid supports test execution on Android devices only, future versions will add support for iOS and other platforms.

4.1. Approach and Architecture

All components of the NeoDroid automation solution are based on the Microsoft Excel technology for data storage as well as on custom code to implement the core automation functionality. Additionally, the solution makes use of the Robotium framework to execute and monitor the actual test execution.

Figure 4 illustrates how NeoDroid has been designed. It comprises the central NeoDroid core and test key file components as well as accessory components such as test data and the test object repository. These components are used to generate the actual test package, which is then executed on the Android device. NeoDroid's architecture follows a specific approach which can be best characterised by discussing four core aspects: test specification, test execution, test data management and object management.

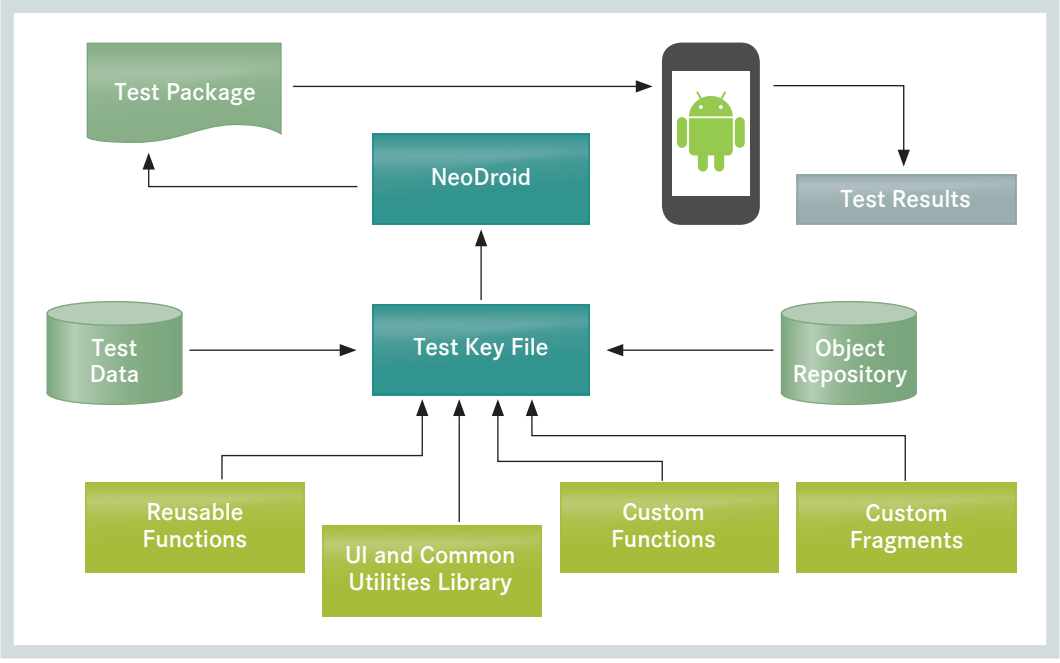


Figure 4: NeoDroid Architecture

4.1.1. Test Specification

During the test specification phase, the foundation for a well-structured, maintainable and efficient test suite is laid. Therefore, tool support for systematic test suite design and easy implementation of sustainable test assets must be included into the tool's design.

In the case of NeoDroid, this has been accomplished by establishing a four-step test specification workflow:

5.1.1.1. Test Generator

Each Test Case consists of a file named 'Key.xlsm', which lets the user select controls (Edit Box, Button, Drop-Down List, etc.) and a corresponding action (Type, Click, Select, etc.) (cf. Figure 5). Separate data sheets and object repositories are maintained that can be referenced while specifying the controls and the data to be passed onto them as input. All that the user is required to do is create a test case by selecting a control-action pair from pre-populated lists, abstracting him from the actual script code. NeoDroid also lets the user call pre-written user defined custom code functions and fragments to tackle any complex requirements during the course of test case designing. When test execution is initiated, NeoDroid automatically generates the code based on the above selections and uploads it onto the device to be run.

5.1.1.2. Test Mapping

Any medium to large size test suite contains a number of test cases which are interdependent. They need to be executed in a particular order so that the necessary flow of execution is maintained. NeoDroid lets you create these parent-child test mappings and warns you if you try to proceed without first executing the parent test cases in the test suite.

Figure 6 shows how test cases are mapped to form a coherent test suite.

Key	Action	Identification Parameter(Text/Index)	Input Value	Output Value
EDITTEXT	TYPE_IN	1	admin	
EDITTEXT	TYPE_IN	r_PasswordBox	p_myPassword	

reUse_Login
reUse_Trial
call_Custom_Action
call_Custom_fragment
ACTIVITY
AUTOCOMPLETETEXTVIEW
BUTTON
CHECKBOX

Figure 5: Test case specification

Execution Trigger	Choose Device	Android API Level	Run @ Brightness Level	Test Case	AUT Namespace	Run Comments
No Run	droid4.0.3	15	10%	TC0001_TestCase1	com.neodroid.presentation.	PresentationAppActivity
Run	droid4.0.3	15	10%	TC0002_TestCase2	com.neodroid.presentation.	PresentationAppActivity
No Run	Android4.0.3	15	10%	TC0001_TestCase1	com.neodroid.presentation.	PresentationAppActivity
No Run	Android4.0.3	15	10%	TC0002_TestCase2	com.neodroid.presentation.	PresentationAppActivity
No Run	Android4.0.3	15	10%	TC0002_TestCase2	com.neodroid.presentation.	PresentationAppActivity
No Run	Android4.0.3	15	10%	TC0003_TestCase3	com.neodroid.presentation.	PresentationAppActivity
No Run	Android4.0.3	15	100%	TC0004_TestCase4	com.neodroid.presentation.	PresentationAppActivity

Figure 6: Mapping of test cases

5.1.1.3. Test Suite Creation

NeoDroid lets you create and manage test suites. You can:

- Prioritise test cases by moving them up or down in the sequence of execution
- Decide how many iterations each test case needs to go through
- Specify different levels of priority for different test cases in the suite
- Have different iterations run with different sets of data for the same test case in the test suite

5.1.1.4. Test Batch Scheduling

With many regression or sanity/smoke test suites, the need arises to run the test suite early in the morning or late in the day, outside office hours. NeoDroid is able to schedule test runs, be it on the device or on the emulator, so that they have been executed during the night and their results are ready first thing in the morning.

4.1.2. Test Execution

Test Execution itself can be controlled from the NeoDroid application. In order to execute a test suite a test data package is generated and transferred to the test environment. Environments may be assembled from physical Android devices or from device emulators such as those found in the Android developer's toolkit (cf. Figure 7).

Once the test data package has been transferred to the test environment, actual execution of the test suite can commence. NeoDroid supports all versions of the Android mobile operating system supported by Google so that mobile applications can be tested thoroughly on a broad range of target platforms.

After the automated execution of the test suite has concluded, NeoDroid collates the individual results into a comprehensive report that can be used to evaluate the test run. Figure 8 shows the contents of a results report.

4.1.3. Test Data Management

Test Automation helps improve the accuracy and reliability of the application by testing with multiple data inputs. For test cases to be extremely efficient it is important to streamline the data input such that the tests can be executed repeatedly with similar – but different – data sets.

Any automation framework relies on abstraction in order to implement data variability in a maintainable

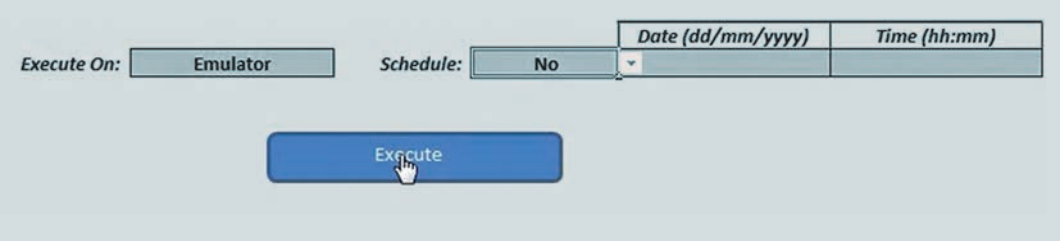
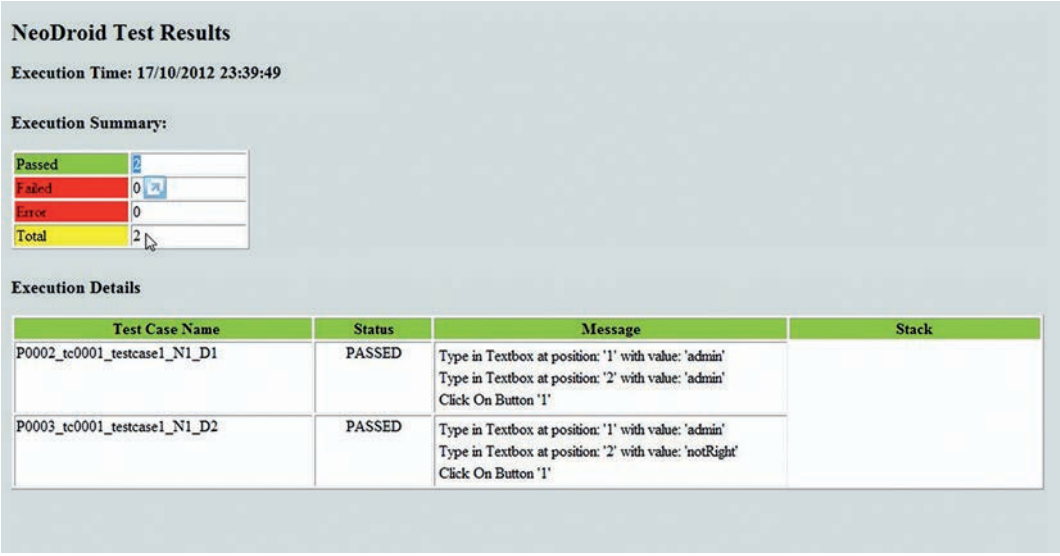


Figure 7: Starting the test execution



NeoDroid Test Results

Execution Time: 17/10/2012 23:39:49

Execution Summary:

Passed	2
Failed	0
Error	0
Total	2

Execution Details

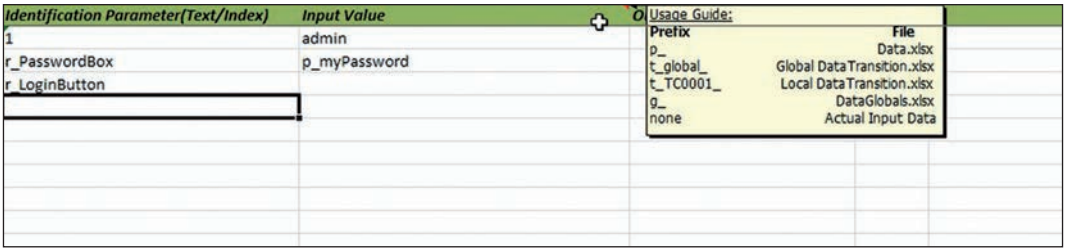
Test Case Name	Status	Message	Stack
P0002_tc0001_testcase1_N1_D1	PASSED	Type in Textbox at position: '1' with value: 'admin' Type in Textbox at position: '2' with value: 'admin' Click On Button '1'	
P0003_tc0001_testcase1_N1_D2	PASSED	Type in Textbox at position: '1' with value: 'admin' Type in Textbox at position: '2' with value: 'notRight' Click On Button '1'	

Figure 8: NeoDroid test results report

fashion. Data abstraction is key to maintainability because it helps manage data usage across multiple test cases in an efficient manner, such that the integrity of the data is maintained at all times. NeoDroid maintains a Microsoft Excel-based structure for test data management which contains the actual data as well as all parameters that reference it (cf. Figure 9).

NeoDroid offers the use of test data on multiple levels:

1. Direct input: Here the data/parameter is passed directly or is hard coded in the test. This does not allow multiple data inputs, and can be used in a scenario where there is no need for multiple data sets to be provided for testing.
2. Transient Data: NeoDroid provides a feature wherein test cases can share data created during the process. If, for example, test 1 creates a certain data value which needs to be referred to in subsequent tests, this is possible using transient data.



Identification Parameter(Text/Index)	*Input Value*
1	admin
r_PasswordBox	p_myPassword
r_LoginButton	

Usage Guide:

Prefix	File
p_	Data.xlsx
t_global_	Global DataTransition.xlsx
t_TC0001_	Local DataTransition.xlsx
g_	DataGlobals.xlsx
none	Actual Input Data

Figure 9: Selection of test data

3. Global Data: It is usually the case that tests have a lot of data which remains common across a suite. To facilitate this and to reduce the redundancy of data usage, NeoDroid lets the user create a test data sheet on a global level. Once the global data has been created, it can be referred to easily in all the test cases. An ideal example would be the login functionality. So the username and password, most commonly used across tests, can be specified as global data.
4. Local Data: Apart from the cases mentioned above, tests also have the facility to use and maintain their own individual test data sets which remain local to them.

At the same time test data from the sources discussed above can be included or excluded from test iterations in a flexible fashion; each row of data can be enabled or disabled to decide whether or not a specific test is executed using the test data entered.

4.1.4. Object Management

In the context of test automation, all the application's interface elements that the test automation suite will interact with need to be known to the test automation suite. In the case of mobile test automation this includes all user interface elements that will need to be used during test execution. The following figure shows a screenshot of an Android app with three user interface elements: A text input box and two buttons, labelled "Search" and "Clear".

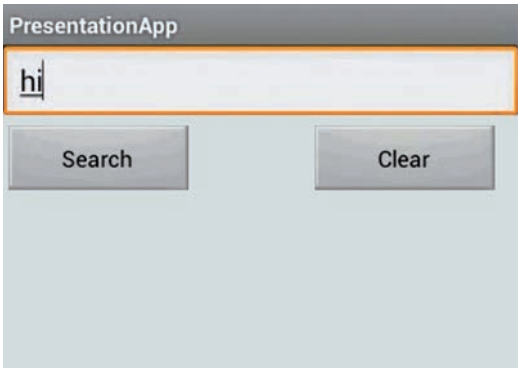


Figure 10: Android user interface elements (i.e. objects)

In order to specify these user interface elements, a so-called test object is defined within the test automation tool that defines precisely how to interact with any given UI element. Such an object repository allows the storing of all the objects that will be used in the scripts in a dedicated facility rather than having to hard code them all over the test scripts. Subsequently, this concept is in widespread use across most test automation suites on the market. This is due to the fact that if the objects are used as hard coded values across the scripts, maintaining the test suite to reflect changes to the user interface of an application or the operating system beneath it would be very difficult and expensive.

In general, the automation tools that provide Record/Playback facility can generate an Object Repository automatically during recording and this repository can be shared/maintained for the entire automation suite. NeoDroid facilitates this auto-generation of the repository by providing a mechanism to the user to create a simple test case to navigate through all the screens of the Android application under test. While traversing all screens the tool grabs all the objects present in the application along with their properties.

Reliable identification of user interface objects often proves to be the most painful part of automation, especially when, after a while, both the application under test and the automation suite have matured and development focus moves to changing existing functionality instead of implementing new features. Unfortunately, this often leads to changes to the order or titles of user interface objects, in turn requiring tedious and complex maintenance of the test automation assets. In order to reduce this risk, support for additional properties like ID, Hint, X/Y Coordinates is implemented in NeoDroid for a stable identification of objects.

A second important aspect of test object management in test automation is the proper use of modularisation in order to restrict changes to as few locations as possible. In order to provide such modularity, NeoDroid supports two types of object repositories i.e. a Global Object Repository and a Local Object Repository:

- A Global Object Repository holds definitions of the objects that can be referred to across the test suite by all the test cases. Users can refer to the objects in the Global Object Repository by prefixing “r_” to the object name.
- In case there are any Test Case specific objects (that are not referred to by any other test case), they can be stored in a Local Object Repository. These objects can only be used by the specific test case for which they are defined. Local objects can be referred to in a test case by prefix “l_”.

4.1.5. Other Considerations

Maintainability of the framework is an important aspect to be looked into. Google may, from time to time, introduce new types of controls, actions or gestures with evolving technology. An accurate mapping of these controls and actions and continuous support for them by NeoDroid is an obvious requirement. Also, given the fact that each Excel containing the test steps or the object repository contains a piece of VBA code in the background, it is very important that we have a central code location from where these can be modified in case of any changes to features supported by NeoDroid. To ensure this, single ‘SRC’ or source files are maintained. Even if hundreds of test cases exist, each test case file picks up its code from these source files. Any change in SRC files propagates itself to all the test cases. This guarantees that updates to the tool remain possible in the future.

Any keyword-based automation framework relies on a decent usability concerning a wide – and at times heterogeneous – audience. NeoDroid, based on MS Excel, is extremely easy to learn. As the majority of PC users have some degree of familiarity with Excel, they do not have an overhead of learning a new tool. NeoDroid has been designed in such a manner that the end user, with very minimal knowledge of Android, can proceed to create vast test suites, without knowing the underlying Java/Android technology. The individual files for the various sections such as test data, key, or object repository have the necessary amount of online help to support the user in providing the right input.

Performance of the device and the effect of the automation on the same, are valuable KPIs which can be put to good use in mobile app testing. NeoDroid is able to monitor and check the performance of battery and memory at any point in the test by just invoking the keywords at the expected points of the test. The results report provides a separate column where the performance indicators for tests using these keywords are displayed. The memory stack details include the peak memory, total usage etc.

4.2. Benefits

NeoDroid provides a range of benefits over pre-existing automation frameworks for Android-based mobile devices. Its strengths lie in the powerful keyword-based automation approach that ensures reliable and stable identification of user interface elements, the modular approach to maintenance of test data, test cases and test objects and its ease of use that provides such powerful features in a relatively simple to use package, enabling users with little test automation expertise to support the test automation effort in collaboration with test automation experts.

4.3. Next Steps

While even today NeoDroid provides a number of tangible benefits already, there are some obvious next steps that need to be taken to take the tool to the next level. All involve extending the reach of NeoDroid to cover additional use cases.

On the Android platform itself, NeoDroid can automate the execution of native Android apps that make use of the rich set of user interface elements present on the Android platform. Non-native apps such as HTML5-based web applications or apps that ignore the standard widget set in order to draw the user interface elements themselves cannot benefit from this approach. For these app types, interfaces will be built into NeoDroid that allow the identification and use of such user interface elements for automation.

A similar issue presents itself when looking at typical mobile app development projects: More often than not, an app is intended to be deployed on multiple platforms in order to cover as big a share of the market as possible. Technologically, this means that apps need to be developed – and tested – not only for Android but often for Android, Apple's iOS, BlackBerry and Microsoft Windows Phone. A next-generation mobile test automation solution will need to provide seamless support for more than one of these platforms to remain compatible. As Android and iOS dominate the market, providing support for these two platforms can be considered a reasonable next step for NeoDroid.

5. Case Study: SQS TC

SQS TC is a second solution used in the field of mobile automation testing, also based on the same automation framework. It is designed around the tenet of providing an efficient, simple and user-friendly interface which is aimed at helping generate and execute mobile application test cases in a simpler way.

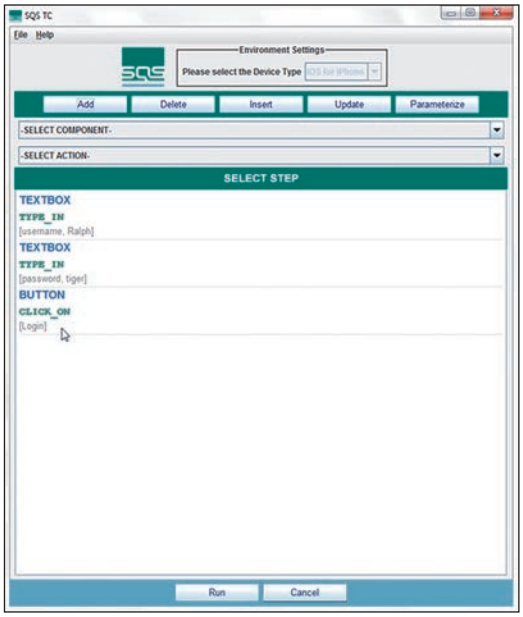


Figure 11: Screenshot of SQS TC on a PC

5.1. Approach and Architecture

The main objective for SQS TC was to develop a framework with a number of utilities supporting mobile automation testing for a number of mobile operating systems such as Android and iOS. The first release of the framework targeted Android while support for the Apple iOS platform was added with the second release.

The following approach defined the implementation of the framework:

1. **GUI Element Coverage**
   The very first thing to do in framework development is to identify the GUI components which are used in mobile applications. So identification of all GUI components and their respective actions are identified as a first requirement for automation framework development.

2. **Lightweight Framework**
   This framework is a lightweight framework i.e. it does not depend on additional components or technologies so that it is easy to implement and deploy for users. Once the executable of the AUT is ready, test automation can commence immediately. Due to this light-weight approach, the framework can be used to test all types of mobile apps.

3. **Platform API Independence**
   In the mobile market OS releases for the major platforms occur very frequently, so there is an immediate need to develop a framework that does not need to change fully with every OS release. When OS releases add GUI elements, the list of automatable elements grows, but existing test cases may be used without change.

4. **Simple, portable data formats**
   As SQS TC is intended to be as independent of platform and platform versions as possible, a portable, yet simple data format to store test cases and associated data was chosen. Considering the same, as well as other benefits such as simplicity, support for data comparison and aggregation as well as programming language support, XML was chosen as a basis for test case files.

5. **Test case generation frontend**
   When targeting small-scale mobile apps, it is imperative to provide a user interface that is flexible and easy to use in order to accommodate team members that may not be test automation specialists. Well-known commercial tools provide such a capability by way of capture-replay-tools which record the manual test execution and generate a test automation script. SQS TC provides such an interface through a small application that may run on a desktop PC as well as directly on the device. This application allows the user to specify test cases by selecting the respective user interface elements, the actions to be performed and the test data to be used (cf. Figure 12).

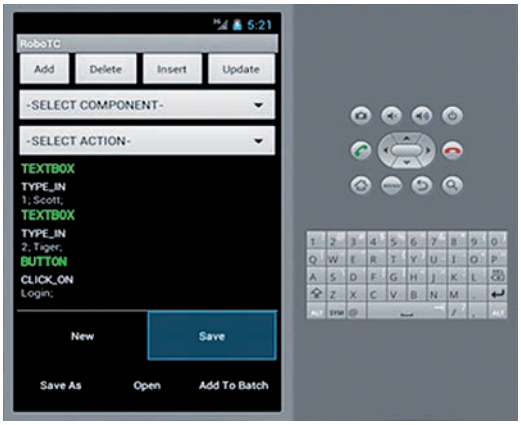


Figure 12: Test case generation on an Android emulator

6. **Test execution on emulated, virtual as well as real devices**
   A key problem when considering mobile test automation is the fact that there is such a huge variety of platforms and devices on the market. For the Android platform alone there is a huge number of different phones and tablets, all of which having unique features such as screen size, resolution, aspect ratio, etc. Therefore, the notion of being able to execute automated test cases on all potential target devices must be accepted as impossible. In the real world, a healthy mix of emulated, virtualised and physical devices prevails, with importance and risk dictating which devices are acquired as a physical device, which are sourced from a cloud provider and which are only emulated. SQS TC was designed to be compatible with physical, virtual and emulated Android devices, so that an effective and efficient risk-driven test management approach is supported by the test automation tool chain.

7. **Compatibility with iOS platform**
   SQS TC was designed with cross-platform portability in mind. However, the actual platforms differ wildly in their use of programming languages, libraries and APIs. Despite this, it is still possible to provide an abstraction layer across multiple platforms by focusing on the user interface elements. Each platform provides a similar set of UI elements, so that only the underlying mechanics of interacting with these UI elements needs to be adapted when porting to different platforms. Currently SQS TC provides implementations for Android and for Apple iOS, the iOS port being written in the native Objective C language. The test case specification app is implemented in a platform-independent fashion, so that this component of SQS TC works not only on Android and iOS but also on potential future platforms.

8. Flexible report generation

A key capability of any test automation tool is the ability to provide precise and useful test reports. As every organisation has slightly different preferences concerning format and contents of the report, the reporting engine of SQS TC is kept flexible. Thus, SQS TC is able to generate reports in a wide variety of formats. It is also able to export reports in a range of document types such as txt, doc, xls, xml and html (cf. Figure 13). Reports contain information such as test case name, date of execution and time of execution, the status of every executed action, i.e. which action has been performed on which component and the status of the action ('passed' or 'failed'). If execution has failed detailed failure information is included.

```
------------------------------------------------------------
Test Case Details
------------------------------------------------------------
TestCase Name      :GNotesTestCase
Date of Execution :27-7-2012
Time of Execution: 14:23:1

------------------------------------------------------------
Execution Details
------------------------------------------------------------

PASSED: Click on Label text 'All'
PASSED: Click on Label text 'Folder'
PASSED: Click On ImageButton at position: '2'

-------------------VERIFICATION POINT---------------------
VERIFIED: 'Backup Success' Dialog exists
------------------------------------------------------------

PASSED: Click On Button 'View'
PASSED: Wait for: '1' duration
PASSED: Go back key pressed
PASSED: Click On Checkbox at position: '1'
PASSED: Click On Button 'Restore'

-------------------VERIFICATION POINT---------------------
VERIFIED: 'Restore' Dialog exists
------------------------------------------------------------

PASSED: Click On Checkbox at position: '1'
PASSED: Click On Button 'OK'
All Verification Points passed; Test Case passed successfully
```

Figure 13: Sample pure-text test report

The implementation of the SQS TC approach is based on the general automation framework and a number of helper components that implement the test case specification tool as well as the test execution engine.

Figure 14 illustrates this architecture. Executable.vbs provides the interface to the real or virtual mobile device and orchestrates the execution of test cases. The test cases, in turn, are managed in the context of test projects which link test cases with test data, the code to implement them on the device and the generation of test reports.

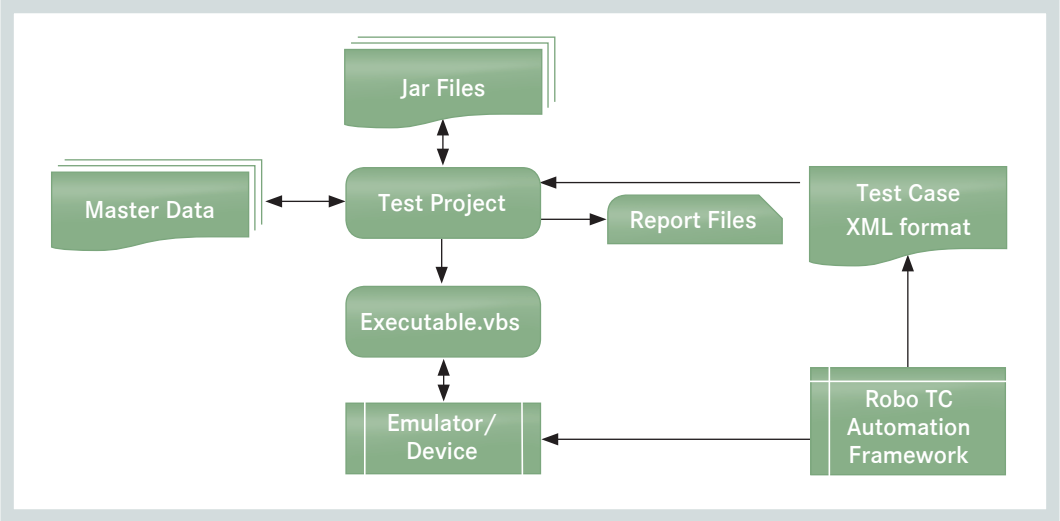


Figure 14: The SQS TC architecture

For the user, this architecture results in the workflow depicted in Figure 15. Starting with a requirements specification (e.g. an SRS or an agile user story), the user responsible for the creation of automated test cases uses the test case generator component to specify the test cases. These test cases are created and stored in XML documents.

Test execution is controlled by the user from the SQS TC tool who commences the generation of executable code and transfer to the device by clicking a single button. After the test run has ended, SQS TC generates the specified test report and – thus – concludes testing.

5.2. Benefits

The approach chosen for SQS TC has shown a number of benefits in real world use, among them:

- Auto generation of test scripts from graphically selected actions. This approach does not require the user to understand implementation details or programming languages
- Support for multiple devices and multiple platforms. The tool can run on any number of physical, virtual or emulated devices and supports Android as well as Apple iOS
- Easy test case maintenance. The test case specification tool allows for easy maintenance of existing test cases. As the tool does not rely on script code, changes are easy to implement
- Flexible reporting – txt, XML, HTML, xls, etc. Report style and format are configurable to fit project specifics
- Run-time binding to GUI components. This renders the test cases immune to smaller changes introduced by new mobile platform versions

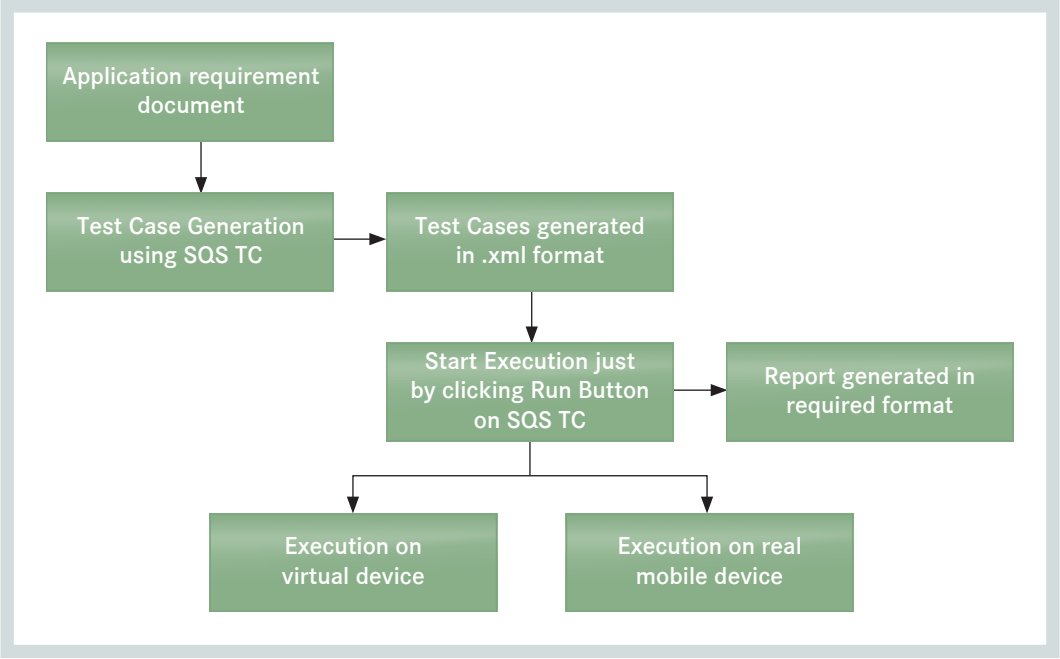


Figure 15: Test execution workflow

- Batch Execution. The tool is able to batch test execution runs for convenience and greater efficiency

SQS TC has been used in a number of test automation projects, where light-weight, speedy automation of small-scale mobile apps was a key requirement.

The automation of a mobile public transport information app highlights the key benefits of SQS TC:

- The app provides features to find local trains, buses etc
- SQS TC was used to automate test cases, such as entering two stations (departure and destination), request the time schedule for public transport connections between the two stations and then checking each connection, even on the displayed map
- The scenarios were automated quickly and successfully using SQS TC

5.3. Next Steps

While SQS TC already covers a wide range of mobile automation use cases, some capabilities are not yet available. One area for improvement concerns the emulation of user interactions through multi-touch gestures.

Secondly, apps that are based on frameworks such as Adobe Flash are chronically problematic to automate, due to the fact that they often do not use standard user interface elements but rather provide their own set of elements. As these are not understood by the underlying mobile operating systems, SQS TC cannot recognise them either.

6. Conclusion

Both NeoDroid and SQS TC show that mobile test automation can be implemented efficiently and with a high return on investment. However, the two case studies also show that it is imperative to understand the concrete requirements, as no single tool would have been able to satisfy both use cases at once without making painful sacrifices.

Therefore, this white paper aims to illustrate that before engaging in mobile automation, every organisation is well-advised to develop a good understanding of its context and requirements and select the test automation solution that fits best. In a young market like mobile test automation, the established players from similar markets (i.e. desktop or server side test automation) are often not the drivers of innovation but are lagging behind smaller, more innovative solutions. There is an old adage that "Nobody ever got fired for buying IBM", hinting at the value of opting for conservative solutions; but in new markets such as mobile, this adage may not be the best advice.

7. Bibliographical References

androidguys. (April 2012). Tablet Sales to double in 2012. Retrieved from AndroidGuys: http://www.androidguys.com/2012/04/10/gartner-tablet-sales-to-double-in-2012/

Canalys. (3. 2 2012). Smartphones overtake Client PCs in 2011. Retrieved from http://www.canalys.com/newsroom/smart-phones-overtake-client-pcs-2011

Gartner Inc. (n.d.). Mobile Market Q3/2011. Retrieved 5. March 2012 from http://www.gartner.com/it/page.jsp?id=1848514

Geronimo, F. (2013). How the latest trends are shaping the Western European mobile phone market.

Glossary Working Party from the International Software Testing Qualifications Board (December 2007). Standard glossary of terms used in Software Testing. Homepage: ISTQB.

Jacob, J., & Tharakan, M. (2012). Roadblocks and their workaround while testing Mobile Applications. testing experience, 8-17.

VisionMobile Ltd. (2012). Cross-platform development tools 2012. London.

Wikipedia. (2013). Android Release Cycles. Retrieved from http://en.wikipedia.org/wiki/Android_version_history