Aiman Mamdouh Ahmad Ayyal Awwad, MSc

Automated Bidirectional Languages Localization Testing for Android Apps Development

DOCTORAL THESIS

to achieve the university degree of
Doktor der technischen Wissenschaften

submitted to

Graz University of Technology

Supervisor

Univ.-Prof.Dipl.-Ing.Dr.techn.Wolfgang Slany

Institute of Software Technology

Graz, August 2017

AFFIDAVIT

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present doctoral thesis.

Date Signature

German Abstract

Mobile Anwendungen werden in vielen Bereichen, wie Wirtschaft, Spiele, Kommunikation, Fotobearbeitung und soziale Netzwerke entwickelt. Die Distribution solcher Apps auf weltweiter Ebene erfordert, dass sie in vielen Sprachen verfügbar gemacht werden, einschließlich bidirektionaler Sprachen für z.B. die arabischsprachige Welt. Erfolgreiche Apps werden in der Regel über einen längeren Zeitraum mit häufigen Releases von neuen öffentlichen Versionen entwickelt, wobei die Übersetzungen von Texten unabhängig von Übersetzern hinzugefügt werden, weil Entwickler nicht über das nötige Wissen verfügen, um ihre Anwendung auf Sprachen zu lokalisieren, die sie nicht selbst beherrschen.

Catrobat ist eine kostenlose und offene visuelle Programmierumgebung für Smartphones. Sie wurde für pädagogische Zwecke entwickelt, um SchülerInnen dabei zu helfen, sich Programmierkenntnisse anzueignen. Sie ermöglicht den SchülerInnen, eigene Animationen und Spiele für ihre Schulfächer zu erstellen, und auch drahtlos externe Hardware zu kontrollieren. Catrobat kommuniziert mit den Jugendlichen in ihrer Muttersprache und ermöglicht es ihnen, die beste Erfahrung in der Sprache ihrer Wahl zu machen.

In dieser Dissertation lokalisieren wir Catrobat in bidirektionale Sprachen wie Arabisch, Persisch, Urdu und Hebräisch und stellen die herausfordernden Aspekte der Lokalisierung in solche Sprachen vor. Darüber hinaus diskutieren wir die Implementierung von Google Material Design Richtlinien, Internationalisierung und Lokalisierung für mobile Anwendungen angewandt auf Pocket Paint, eine Android Mal-Anwendung. Das angestrebte Ziel dieses Redesigns ist es, die Benutzerbasis zu erweitern, indem sie die Gesamtnutzbarkeit verbessert und von rechts-nach-links geschriebene Sprachen wie Arabisch unterstützt. Die wichtigsten Herausforderungen des Redesigns sind die Feinheiten, die sowohl rechts-nach-links- als auch links-nach-rechts-Sprachen betreffen, z.B. die Positionierung, Übersetzung, die Spiegelung von Text und grafischen Elementen, das “Wann” und “Wann nicht”, bzw. ob oder nicht gespiegelt werden soll. Im Zusammenhang mit der Übereinstimmung mit den Material Design Guidelines haben wir mit sechs Benutzern im Alter von 13 Jahren unserer Zielgruppe einen User Experience Test durchgeführt. Alle Teilnehmer bewerteten die neu gestaltete Anwendung einfacher, attraktiver und intuitiver im Vergleich zur Vorgängerversion.

Während automatische Tests prinzipiell und im Allgemeinen wichtig ist, um in der Lage zu sein, qualitativ hochwertige Software zu entwickeln und viele Arten von Problemen zu lindern, die sich häufig in Softwareprojekten ergeben, werden solche automatischen Tests absolut notwendig, wenn Entwickler, die nicht über ausreichende Kenntnisse über rechts-nach-links geschriebene Sprachen besitzen, Code weiterentwickeln müssen, der in der Vergangenheit auch für bidirektionale Sprachen geschrieben wurde. Es gibt ein paar bidirektionale Lokalisierungstests für mobile Anwendungen. Allerdings ist ihre Funktionalität begrenzt, da sie nur Übersetzungen und Annahme von Gebietsschemata abdecken.

Deshalb stellen wir unseren Ansatz für die Automatisierung der Tests für die Lokalisierung mit bidirektionalen Sprachen von Android-Anwendungen vor. Das Hauptziel des vorgeschlagenen Ansatzes ist es, die Besonderheiten der verschiedenen Lokalisierungsanforderungen zu dokumentieren und auf Lokalisierungsdefekte im Produkt zu prüfen. Die vorgeschlagenen Methoden werden verwendet, um Fragen von bidirektionalen Apps im Allgemeinen und speziell für die arabische Sprache zu testen. Die Ergebnisse zeigen, dass die vorgeschlagenen Methoden in der Lage sind, Mängel in App-Designs zu enthüllen. Darüber hinaus haben die Methoden die Fähigkeit, sicherzustellen, dass die lokalisierte App den geplanten Spezifikationen und allen Erwartungen der lokalen Endbenutzer entspricht und garantieren, dass das Produkt kulturell kongruent gegenüber arabischen lokalen Konventionen ist. Weiters führt das automatisierte Testen der Besonderheiten in Bezug auf Lokalisierungen zu einer signifikanten Abnahme des Aufwands und der Zeit für die Qualitätssicherung und Regressionstests.

Abstract

Mobile applications are getting developed in many areas, such as business, gaming, communication, photo editing, and engaging in social networking. The release of such apps on a worldwide scale requires them to be made available in many languages, including bidirectional languages for, e.g., the Arab speaking world. Successful apps are usually developed over a long period of time with frequent releases of new public versions, and languages are added independently by translators because developers do not possess the knowledge needed to localize their application to languages they do not speak.

Catrobat is an open source visual programming environment for smartphones. It has been developed for educational purposes to help students visualize and understand learning material. It allows students to build their own animations and games for their classes in academic subjects and wirelessly control external hardware. Catrobat needs to talk to the young children at schools in their native language and to enable them to get the best experience in the language of their choice.

In this thesis, we localize Catrobat into bidirectional languages such as Arabic, Persian, Urdu, and Hebrew, and introduce the challenging aspects of localization to such languages. Also, we discuss the implementation of Google's Material Design guidelines, internationalization, and localization for mobile applications in the case of Pocket Paint, an Android painting application. The intended goal of this redesign is to broaden the user base by improving overall usability and supporting right-to-left written languages such as Arabic. The main challenges of the redesign are the intricacies to thoroughly support both right-to-left and left-to-right scripts, e.g., the positioning, translation, mirroring of text and graphical elements, the 'when' and 'when not' to mirror. Related to the Material Design guideline compliance we carried out a user experience test with six users (age 13) of our target group. All participants rated the redesigned application being simpler, more appealing and concise in comparison to the previous version.

While automatic testing by itself is important, in general, in order to be able to develop high quality software and alleviate many kinds of problems commonly arising in software projects, such automatic tests become absolutely essential when developers that do not possess enough knowledge about right-to-left languages need to maintain code that is written for bidirectional languages. A few bidirectional localization tests of mobile applications exist. However, their functionality is limited since they only cover translations and adoption of locales.

Therefore, we present our approach for automating the bidirectional languages localization testing process for Android applications. The main objective of the proposed approach is to document the peculiarities of different localization requirements and check for any localization defects in the product. The proposed methods are used to test issues of bidirectional apps in general and specifically for the Arabic language. The results show that the proposed methods are capable to reveal deficiencies in the app's design. In addition, the methods have the ability to ensure that the localized app matches the intended specifications and all expectations of local end users, and guarantee the product is culturally congruent to Arabian local conventions. Further, automated localization testing leads to a significant decrease in effort and time spent for quality assurance and regression testing.

Publications

1. Chapter 11 introduces the main challenges encountered in localizing apps into bidirectional languages, provides all the challenging aspects of localizing Catrobat into bidirectional languages and achieve comprehensive solutions to the many and correlated challenges presented by such languages. The paper was published in the journal as follows:
   Aiman M. Ayyal Awwad, "Localization to Bidirectional Languages for a Visual Programming Environment on Smartphones," International Journal of Computer Science Issues, Vol. 14, Issue 3, May 2017. doi:10.20943/01201703.113

2. Chapter 12 discusses the implementation of Google's Material Design guidelines, internationalization, and localization for mobile applications in the case of Pocket Paint, an Android painting application. A paper of the results of Chapter 12 has been accepted for publication in the conference proceedings as follows:
   Aiman M. Ayyal Awwad, Christian Schindler, Kirshan Luhana, Zulfiqar Ali, Bernadette Spieler, "Improving Pocket Paint Usability via Material Design Compliance and Supporting Internationalization and Localization on Application Level," 19th International Conference on Human-Computer Interaction with Mobile Devices and Services, 4th-7th September, 2017, Vienna, Austria. doi:10.1145/3098279.3122142

3. Chapter 13 presents our approach for automating the bidirectional localization testing for Android applications with a complete consideration for BiDi-languages issues. The paper was published in the journal as follows:
   Aiman M. Ayyal Awwad and Wolfgang Slany, "Automated Bidirectional Languages Localization Testing for Android Apps with Rich GUI," Mobile Information Systems, April, 2016. doi:10.1155/2016/2872067

Ph.D. Funding

For this Ph.D., a three-year grant was given from the Erasmus Mundus AVEMPACE III. As a result, the research activities for the three-year period have been carried out at the Institute of Software Technology, Graz University of Technology, Graz, Austria.

To the memory of my parents, Umm Ahmad and Abu Ahmad
May Allah rest their souls in Paradise.

To my wonderful wife . . . Wala'

To my son: Omar

To my daughter: Sarah

To my brothers: Ahmad and Haitham

I dedicate this work.

Acknowledgments

In the Name of Allah, the Most Gracious, and the Most Merciful
All praise and thanks are due to the Almighty Allah who always guides me to the right path and has helped me to complete this dissertation.

It is my pleasure to thank from the heart, a number of people who helped me and contributed to the completion of my thesis. Without their support and motivation, I could never have accomplished any of my work.

First and foremost I would like to record my gratitude to my supervisor, Prof. Wolfgang Slany, for his great supervision, consistent support, valuable advice and perceptive suggestions throughout the development of this dissertation. I appreciate all of his special ways, directions and advice to complete this dissertation and enable me to open my mind to challenge myself. Really I never forget Prof. Slany support.

I would like to thank my family. My ever-supportive wife - Wala'. Who has been with me from the very beginning of this work, and whose love and kindness have helped me reach this far. Thank you for your great support and encouragement in my Ph.D. journey.

I'm particularly grateful to my external reviewer, Dr. Ziad Salem, for his invaluable suggestions for improvement. Many thanks Dr. Ziad.

I also wish to extend my thanks to Dr. Christian Schindler, for his support throughout my Ph.D. He always supported me with his valuable discussions and guidance. Many thanks, Dr. Schindler.

I could not have reached so far without the support and the affection of my friends: Shaher Zyoud from Palestine, Amr Almaktry from Yemen, Nur El-din El-Rez from Syria, Dr. Ahmad Mazen from Egypt, Dr. Mohammed Boudjada from Algeria, and Muhammad Umar from Pakistan. I consider myself lucky to have met such special people: you and our memories will always remain in my heart.

Best thanks to Catrobat development team, especially, Bernadette Spieler, Anja Petri, and Annemarie Harzl.

I would like to thank all my Graz University of Technology friends for their best wishes, help, and support.

Finally, this research was supported by the Erasmus Mundus AVEMPACE III scholarship.

Aiman M. Ayyal Awwad

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

This chapter introduces key components of this study. It describes the field of research, scope of the work, the problem statement and researcher's challenges and highlights the research statement and contributions explored within the study. It ends with the structure of the thesis outline.

1.2 Background and Motivation of the Thesis

1.2.1 Bidirectional Localization Process

To design effective educational apps, solve basic problems, and to attract inexpert developers to build interactive applications a visual programming environment (VPE) can be used [1]. VPEs let users create programs by manipulating program elements graphically instead of specifying them textually [1][2]. VPEs are very popular in programming education for young children like primary and secondary schools, since they do not require previous knowledge of programming syntax and create an environment where compile-time errors are nonexistent [2]. It is important to investigate the characteristics of VPEs as they used in the programming education for young children. When using VPEs, it is relatively easy for the young children to create simple games since there are usually built-in libraries in the language environments [2][3].

In recent years, many so-called VPEs have been developed for educational objectives. Examples include Scratch[1], Kodu[2], Snap![3], and Catrobat[4] [2][3]. They are not only easy to learn, but also introduce rich and charming visual interfaces. Catrobat is a visual programming language developed for smartphones. Children can easily learn how to program

[1]https://scratch.mit.edu/
[2]https://www.kodugamelab.com/
[3]http://snap.berkeley.edu/
[4]https://www.catrobat.org/

without having to think about drawbacks like compile-time errors or complicated workflows. Inspired by Scratch, Catrobat also defines blocks which can be snapped together in order to form a program. Unlike Scratch, Catrobat programs can be created and executed by entirely using smartphones [3].

Pocket Code, Catrobat's version for Android platform, is a learning application for mobile devices that has been developed in Austria at Graz University of Technology. With Pocket Code children and young people can create their own games, stories, animations, interactive music videos, and many types of other apps, directly on their phones or tablets, and taking advantage of multi-touch, built-in sensors, and the full display resolution of the device. It uses a visual programming language and it is developed by an open source project. Also, it is featured by Google on Google Play for education platform [3][4].

Catrobat lets students to program and design games for their classes in academic subjects such as science, math, and languages by effectively developing and adapting the learning material themselves. On various occasions, children and teenagers were being taught to program their own applications. They had a lot of fun using VPE to build a simple game without any previous knowledge about programming and thus, not just being users but being developers too.

Nowadays, many countries around the world are promoting programming education for primary and secondary schoolchildren [5]. Some countries have already adopted programming as a major subject in primary education. In those countries, almost always VPEs are employed to teach programming to the children [2][5]. Therefore, the release of VPEs on a worldwide scale requires them to be made available in many languages, including bidirectional (BiDi) languages such as Arabic, Persian, Urdu, and Hebrew. This asks for internationalization (I18n) and localization (L10n) of the product. In software development, internationalization and localization are processes of adapting computer software for non-native regions, especially other locales, cultures, and environments [6].

Localization of software for bidirectional languages has still not reached its full potential due to a shortage of research [6]. At present, most of the localization of VPEs is only expressed in language translation and adaptation of time, date, number, and currency formats. There are many challenging aspects of the bidirectional languages which have been neglected during localization. Until recently, many of these aspects were considered non-essential. However, with the wide-spread of the smartphones and the increased use of mobile applications in business, education, gaming, communication, and engaging in social networking among bidirectional languages speaking populations, many of these aspects are becoming necessary, even expected requirements.

The localized VPE should meet a local user's expectations in terms of language, culture, and user experience. However, Catrobat has specific issues that should be considered during the localization process. To enhance the usability of localized smartphone's applications, Google develops the Material Design[5] guidelines for bidirectionality. In these guidelines, UI layout for languages that are read and written from right-to-left, such as Arabic and Urdu, should be mirrored to ensure content is easy to understand [7].

[5]https://material.io/

1.2.2 Bidirectional Localization Testing

The rapid proliferation of smartphones and the fast growth of the internet make it easy for "apps"(applications for mobile devices) to be accessed and downloaded from all over the world [8]. According to the latest preliminary results from International Data Corporation (IDC), in the fourth quarter of 2016, smartphones sales topped in the worldwide with 428.5 million units and at the same time Android was the most popular operating system with market share reach of 86.8 percent [9]. The greatest challenge is not only to develop apps but also to test them to guarantee their usability and robustness. Usually, apps are distributed to many countries and language regions [8][10]. Many apps on Google Play are available in more than 30 and up to 50 different languages from all over the world, including languages with exotic fonts such as for traditional and modern Chinese, Greek, Thai [11]. Every country or region has its own culture and customs; accordingly, Google strongly encourages developers to localize their apps to meet demands of local users and thereby increase sales.

However, usually developers have little knowledge about localization, and even if they would have the knowledge for all the languages they would like to support, high-frequency manual testing of all localized versions is practically infeasible, especially in a highly iterative development process where tests would have to be executed many times a day in order to avoid the accumulation and proliferation of bugs and other problems among members of the development team. Consequently, automatic app localization testing is getting more and more attention [12].

In general, software globalization is a process which has two phases: internationalization, and localization [6]. Internationalization refers to the process of designing and developing a system that supports different languages and regions. Localization refers to the process of modifying internationalized software for use in a specific country, region, or culture, by adding local-specific features and translated text [13][14].

The quality of international software is completely dependent on the software's localization level; therefore, localization testing is an essential part of quality assurance of international software and thus has turned into a major type of international software testing [14][15].

1.3 Problem Statement and Challenges

At present, most of the localization of software is only expressed in language translation and adaptation of time, date, number, and currency formats. Additionally, the localized language accuracy, integrity, interface layout, and document contents must satisfy the demands of local users and regional culture [16]. However, localization testing usually is either not done at all or infrequently done manually and only partially, usually neglecting languages not spoken by the testers; for example, localized versions of apps are only spot-tested manually for a few selected languages before a release.

Popular software often is produced in many language versions in order to be more understandable and usable for users. To an overwhelming percentage, these applications are built for left-to-right (LTR) languages such as English or German. By contrast, the Arabic language is written and read from right-to-left (RTL) [16]. The Arabic language additionally is cursive which means that the letters are connected together like English handwriting. The same character set has been utilized to express more than 25 languages in addition to Arabic [17].

In the area of software localization, the Arabic language is considered as one of the most challenging languages. The Arabic language "differs tremendously in terms of its characters, morphology, and diacritization from other languages, and to claim otherwise would be a mistake" [18]. It is ranked as the 5th language in the world in a number of native speakers, and it is also one of the six official languages of the United Nations [17]. As a result of these, different issues in software localization to the Arabic language should be considered.

Bidirectional languages such as Arabic, Farsi, Urdu, and Hebrew, where text is read and written from RTL while numbers are read and written from LTR, require layout customizations not only for text but also for all user interface (UI) widgets, including buttons, text views, edit texts, seek bars, check boxes, menus, and dialog boxes. There are many UI elements that need to be adjusted in both features and content localization. These include strings, layout, images and multimedia, character sets, and locale data [16].

Software localization testing is a critical method that is carried out to control the quality of a product's localization for a specific geographical region or culture. This test is like a passport for your app to transfer from country to country. The main goal of automatic software L10n testing in a test driven development process is to document the peculiarities of different localization requirements for those developers that do not know about them from their own cultural background and to make sure that bugs and deficiencies that are introduced at a later stage do not break the localization aspects of the product.

Indeed, the tests must have the ability to confirm the functionality and performance of localized software and components according to the original product and to detect linguistic and functional problems [19][20]. It is essential that the correctness of translations is verified and that consideration of culture issues is guaranteed. However, the localization process often introduces severe issues such as the following [21][22]:

i. Clipped strings, or strings that overlap the edge of UI elements on the screen

ii. Date and time formats

iii. Untranslated strings (strings are displayed in the source language instead of their target language, possibly missing translation; this can happen quite frequently when the app is developed in a continuous way and not all translations can be added at the same time)

iv. Incorrect layout or text direction

v. Incorrect alphabetical sorting

Additionally, the L10n tests are in some cases able to reveal deficiencies in the software's design [16][19]. Many localization issues need to be reviewed as discussed by Kopsch [13]. Developers should consider the particular characteristics of each destination. As a result of the translation, there are also some cosmetic checks that need to be implemented. For example, one needs to check that the labels still fit on the screen, whether they were clipped, that they are not overlapped with other UI elements, and so forth as discussed by Cavalleri [14].

During the localization process, many good practices and tips should be considered, and the tester should follow the localization checklist and make it a priority to reveal I18n and L10n defects by concentrating first on five languages including English. Experience has shown that we are most likely to find specific issues in the following languages: German, Japanese, Arabic, and Hindi, as discussed by Kotze [19].

However, localization of mobile apps for BiDi-languages has still not reached its full potential due to a shortage of research. Most of the proposed approaches are designed to test localization issues for desktop and traditional web applications as discussed in [16][20]. In addition, a few of the published papers identify the various unique challenges associated with localization testing of mobile applications as discussed in [12][23] and identify the components to be included in an effective test strategy to build localization testing as discussed in [12].

1.4 Thesis Statement and Contributions

In this thesis, the visual programming environment (Catrobat) is localized into bidirectional in general and specifically to the Arabic language (Arabization), furthermore all the challenging aspects of the bidirectional, which facing the application's developers, are introduced as well as the comprehensive solutions to these challenges are proposed. Without considering these challenges by the software localization specialist, the quality of bidirectional languages product will continue to lag behind other languages.

In the course of usability, firstly, we want to broaden the user base by multi-language and bi-directionality support. We want to implement multi-language and bi-directionality support on application level since previous versions' support was only on operating system (OS) level. From a usability point of view, it is better to be able to set an app's language within the app itself, either default to the language of the OS, or independently. This is particularly true for poorly translated apps. Secondly, we want to comply with Material Design guidelines to improve the graphical user interface. Overall, both directions of redesign should contribute to an improved user experience in the sense of improving look, feel and usability of Pocket Paint.

Moreover, an automated BiDi L10n testing approach for smartphone applications is introduced; we have applied the proposed methods to test critical issues in Catrobat. The main contribution of this thesis is re-engineering Catrobat to localize it with full and complete considerations for BiDi-languages issues, in particular, the Arabic language, and to check the app for any localization defects in the user interface. The bidirectional languages localization testing results show that the product is cosmetically correct, linguistically accurate, and culturally appropriate. Therefore, it meets bidirectional requirements, complies with bidirectionality design guidelines, and can be employed in programming education for young schoolchildren.

1.4.1 Bidirectional Localization Testing in the Mobile World

A tendency toward mobile applications development has increased in the past few years. Mobile phones and apps play an integral part of our life, and applications must work correctly anytime and anywhere. On the developer side, offering localized versions of one's software simply increases the number of potential customers, and there thus is a strong economic motivation to offer one's software in as many major languages of potential markets as possible. However, some kind of internationalization and localization testing, be it manually or automatically, is necessary before distribution to the market makes sense.

The localization testing of apps faces many issues because of the complexity of testing these apps and the limited resources of mobile devices. Localization testing for mobile applications is more complex and challenging as compared to localization testing of traditional web-based and desktop applications [24].

Mobile applications work in multiple operating systems such as Android and iOS. Especially in the case of Android, the many different Android versions supporting totally different GUI elements, the additional GUI modifications by hardware manufacturers such as Samsung or HTC, the huge number of different keyboard apps that users can freely choose from, and the hugely varying display resolutions and aspect ratios of devices, make the development of the GUI of apps for all these combinations a difficult challenge.

Figure 1.1 illustrates the process of designing an effective localization testing approach for BiDi-languages. The key elements that should be considered while following this approach are as follows:

i. The localization testing needs to be done on different types of devices on different platforms.

ii. In addition to the original language, typically US English, the mobile application needs to support BiDi-languages.

iii. The automation of localization testing can play an important factor in reducing the time and cost of testing the application in BiDi-languages.

1.4.2 Localization Test Automation

Manual testing for multilingual mobile apps is time and resource consuming, while automatic testing can save time and effort as well as increase accuracy and repeatability for localization testing. In general, automated testing can be considered the most desirable type of testing. The automated test can also be written to control the progress of system development and to find defects early and efficiently [12][25].

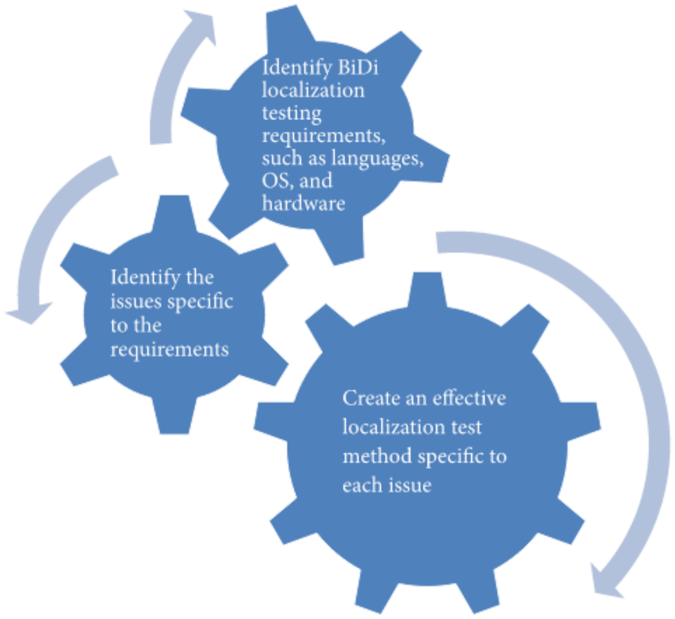

Identify BiDi localization testing requirements, such as languages, OS, and hardware
Identify the issues specific to the requirements
Create an effective localization test method specific to each issue

Figure 1.1: BiDi localization testing process.

It has been claimed that app features that are not tested are in fact not existing, the rationale being that refactoring, which by definition is the improvement of internal software quality without adding new features and a practice that tremendously helps to develop robust software, can only be done under the presence of tests, as otherwise the maxim "never change a running system" would prevail, and refactoring without tests can lead to the unintended elimination of untested functionality or could result in even worse consequences.

On the other hand, when automatic tests can be run by pushing a button and the tests are all "green," that is, there is no unexpected behavior of the software, developers can be sure that the current version of the app conforms to or, in other words, is correct with respect to these test cases. Psychologically, there is the added benefit that the feeling of the developers and, in case they are somehow in the development loop, also the customers is good when they see that the tests were all successful. If all features are tested, the development productivity is increased, among other benefits, as a result of the reduction of time spent for debugging, a time that consequently can then be spent on more productive tasks [26].

Moreover, the automation for localization testing is very effective when test steps are consistently repeated in the same but also in a larger number of testing environments. With the different types of platform and operating systems available for smartphones, it would be impractical to manually test on all permutations [12][26].

1.4.3 Why Test First?

In test driven development, the developer first writes an automated test which initially must fail before writing the corresponding production code. This test describes a desired improvement or new feature and thereby is both a formal specification that can be automatically checked for conformance and a human readable description of a test case. In a second step only does the developer write production code to pass the test, and then the test should run successfully [26].

In the development of our Pocket Paint app, we wrote an automated test that defines the text direction in BiDi-language. Since there was no code yet to make the test pass, this test should initially have failed. However, unexpectedly, the test case passed! How was this possible? The reason, in this case, was that the test case itself was incorrect as it checked the directionality of text incorrectly with the help of a locale configuration, as shown below.

```
final int expected=View.TEXT_DIRECTION_LOCALE;
assertEquals(mView.getTextDirection(),expected);
```

Hence, an immediate feedback was given to the programmer to use a different configuration for BiDi text direction, as shown below. Had the test been written after the production code already existed, the test would also have succeeded, but for the wrong reason!

```
final int expected=View.TEXT_DIRECTION_RTL;
assertEquals(mView.getTextDirection(),expected);
```

The automated tests are written to document software effectively for the developers, whereas the test cases describe the specifications that are defined by the customers, and also to avoid any ambiguity or miscommunication between users and developers. However, if the developer wants to write a test about any subject, he should first understand the subject under test.

One of the best ways for developers to understand the requirements is by translating them into tests. Several advantages for writing the test in advance of the code are as follows:

i. If the tests are written in the last phase (when all coding has been done) it is highly probable that they will not be written.

ii. Developers take more responsibility to produce high quality products.

iii. If the tests are written early, they will be extremely helpful because more of the customer team's needs and expectations are clearly communicated to the developers, and thus there is less possibility for miscommunications.

1.4.4 Why Is Localization Testing for BiDi-Languages so Important?

Localization testing is a type of quality assurance testing; it mainly focuses on the evaluation of the product's functionality, cosmetics, and quality of the localization. The main goals of automatic app localization testing for BiDi-languages are as follows:

i. To document the attributes of different localization issues for those developers who do not know about them from their own cultural background.

ii. To make sure that bugs and deficiencies that are introduced at a later stage do not break the localization aspects of the product.

iii. To detect and report app's localization defects.

1.5 Structure of the Thesis

The rest of the thesis is structured as follows.

- **Chapter 2: Mobile World and Applications**
  In Chapter 2, we explain the mobile device evolution from basic phones to the current smartphones, and we review the history, growth, and usage of smartphones. Furthermore, we summarize the impacts of smartphones on diverse sectors of society. Finally, we present the different mobile app types of mobile apps and discuss the pros and cons for each of them.

- **Chapter 3: Mobile Operating Systems**
  In Chapter 3, we present the major mobile operating systems which are Android, iOS, Windows and Windows Phone. Also, we introduce the different types of App Stores, where apps can be downloaded and reviewed.

- **Chapter 4: Key Challenges in Mobile Application Testing**
  Unlike testing traditional desktop and web applications, the mobile app testing is more complex. This chapter provides the key challenges in mobile application testing such as the customer, device fragmentation, sensors and interfaces, internationalization and localization, and mobile browsers with their possible solutions.

- **Chapter 5: Internationalization and Localization of Software**
  Chapter 5 gives a problem description and a formal definition of the internationalization and localization processes. This includes a presentation of internationalization and localization elements and major cultural and linguistic differences. Furthermore, it explains what kinds of changes have to be made to obtain a localized software application, and presents the collaboration between people involved in such processes. We also try to show why localizing mobile app is a must and so important, and finally, we introduce the importance of localization process in app stores.

- **Chapter 6: Challenges of the Arabic Localization in Mobile World**
  The technical challenges of the Arabic language bring up many questions, which we address in this chapter: first we look at the internet and mobile penetration in the Arab world, then we delve into the characteristics of the Arabic writing system (bidirectionality, character shaping, ligatures, and justification) and how we can handle

its localization. Finally, we give an overview of the differences between Arabic, Farsi, and Hebrew.

- **Chapter 7: Visual Programming Environment for Children: Catrobat**
  In this chapter, we give an overview about Catrobat as well as we list all the programming language elements of Catrobat.

- **Chapter 8: Mobile Automation Test**
  Chapter 8 provides the mobile automation test. This includes a brief discussion on testing types and how to test your app in different types and to make sure that it is robust, stable, usable, and bugs-free as possible. The chapter covers a part of the mobile test environment which includes test equipment: real devices, emulators, and simulators. Furthermore, it discusses the advantages and disadvantages of such equipment.

- **Chapter 9: Background and Related Works**
  In Chapter 9, we provide the theoretical background of internationalization and localization testing and discuss some studies related to these types of testing.

- **Chapter 10: Bidirectional Localization Testing Architecture**
  In this chapter, we present the Robotium framework, its different features, and its benefits for the world of automated testing. Then we give an overview of Espresso testing framework for mobile applications and we discuss the framework's main components which include view matchers, view assertions, and view actions. Finally, we present the tools which are used by the proposed approach such as Jenkins, Git, and Gradle.

- **Chapter 11: Localization to Bidirectional Languages for a Visual Programming Environment on Smartphones**
  In Chapter 11, we introduce the main challenges encountered in localizing apps into bidirectional languages, then we delve into the considerations for localizing bidirectional apps. Finally, we provide all the challenging aspects of localizing Catrobat into bidirectional languages and achieve comprehensive solutions to the many and correlated challenges presented by such languages.

- **Chapter 12: Improving Pocket Paint Usability Via Material Design Compliance and Internationalization and Localization Support on Application Level**
  In Chapter 12, we discuss the implementation of Google's Material Design guidelines, internationalization, and localization for mobile applications in the case of Pocket Paint, an Android painting application. The intended goal of this chapter is to show how the redesign of Pocket Paint could broaden the user base by improving overall usability and supporting right-to-left written languages such as Arabic.

- **Chapter 13: Bidirectional Languages Localization Testing Framework: Implementation and Analysis**
  Chapter 13 presents our approach for automating the bidirectional localization testing for Android applications with a complete consideration for BiDi-languages issues.

The objective is to ensure that the localized product is fully functional, cosmetically correct, linguistically accurate, and culturally appropriate.

- **Chapter 14: Conclusion and Future Work**
  Chapter 14 concludes this thesis and also reflects on future developments related to internationalization and localization processes as well as the usability testing for a multi-language and bidirectional script support on application level.

Chapter 2

Mobile World and Applications

This chapter describes the history of smartphone and primarily focuses on the impact of smartphones on business, education, health sectors, human psychology and social life, and finally presents the different types of mobile app.

2.1 Overview of Smartphones

The word *mobile* originates from the Latin word *mobilis*, which itself is derived from the Latin verb *movere*, “to move”- to be able to move freely and easily. The word “mobile” expresses different meaning in time. A decade ago, the mobile device was symbolized by a laptop or feature phone. Nowadays, laptops are not widely seen as true mobile devices. There is another device that is much more mobile like smartphone [27].

In the real sense, a smartphone is a mobile phone with advanced aspects, operating system, and functionality besides basic phones features like making calls and sending text messages. The smartphones are supplied with the features to display images, play videos, play games, navigation, built-in camera, audio/video playback, and recording, send/receive e-mail, engaging in social networking, web surfing, wireless Internet and much more. In the first place the smartphones were only considered for business use due to their cost and application, but not today, today we are in a rapidly populated smartphone society with the smartphones from several vendors presenting a variety of advanced functionalities and features on a small device [27][28].

With the noticeable growth in the industry, a smartphone has recently achieved to strike a long-expected milestone. Obviously, in the global shipments, smartphone overtook PCs. Today smartphones give customers, promoters, and publishers the ability to better engage and socialize using the present everywhere experience. Due to its unlimited spreading and social acceptance, we can find smartphone in schools, educational institutes, hospitals, public places and shopping centers etc [28].

2.2 History of Smartphones

In 2007, Apple Inc. released the first smartphone called iPhone; the smartphone has a feature of a multi-touch interface, but in reality, the smartphone has been on the market since 1993. The difference between today's smartphone and early smartphone is that early smartphones were designed to be used by corporate customers and for enterprise's communication objectives, and in addition, those phones were too costly especially for the public customers [29].

The smartphone period is divided into three major phases. The first phase was focused on enterprises. During this phase, all the smartphones considered the corporations as their target and the features and functions should follow the corporate requirements. This period began with the first release of the smartphone called 'The Simon' from IBM in 1993. The revolutionary device of this period is Blackberry, it is supplied with many features like email, fax, web surfing, navigation, and built-in camera. The smartphones of this phase were totally built according to the enterprises objectives [28][29].

The second phase of smartphone period started with the release of iPhone. Steve Jobs of Apple unveiled the iPhone which he referred to as a "revolutionary and magical product." The iPhone, the great leap in the smartphone market, was first unveiled in January 2007. This was the period when first time ever company revealed the smartphone targeting the public customers market [28]. The Apple device introduced numerous concepts for a design that have been adopted by modern smartphone platforms, such as the use of multi-touch gestures for browsing.

At the end of 2007, Google introduced Android operating system with the aim to dominate the customer smartphone market. However, Google announced that they will introduce the Android operating system for free and anyone will be able to use and customize it. The company's intention during this period of time was to develop features that the customers need and expect. These include email, social networking, audio/video playing, internet navigation, and chatting. At the same time, Google keeps the cost at a lower level to attract more and more customers [27][29].

The main difference between iOS and Android is the domain of devices. The iOS from Apple is strictly biased to products from Apple. It is used only on iPhone, iPad, and iPod. At the same time, Android runs on many smartphones on the market. The great players like HTC, Samsung, Sony, LG use Android on their smartphone. All in all, you have the ability to reach more users by developing apps for Android.

The third phase of smartphones focused on closing the gap between enterprise view and general customers view. Hence, the two views were mainly united to improve the display resolution, display system technology, mobile operating system, create more long life batteries and enhance the graphical user interface and many more services within these smart devices.

In 2008, this phase started with the revolutionary upgrades in the mobile operating system and during the last eight years, there have been many upgrades in Apple iOS, Android, and Blackberry OS. The most common mobile operating systems (iOS, Android, Blackberry

OS, Windows Mobile) and key smartphone vendors (Apple, Samsung, HTC, Motorola, Nokia, LG, Sony etc.) are focussing on developing new features in operating systems and devices which will introduce an exciting useful feature to enterprise and global customers. The role of Android has been enormous during this time period since it is an open source operating system it introduces a huge opportunity to all vendors to build new devices. Therefore, nowadays, Android is the best-selling smartphone platform [28].

2.3 Smartphones Growth / Usage

Today, smartphones are being produced by numerous vendors and are one of the fastest growing sectors in the technology domain. Operating systems include Google's Android, Apple's iOS, BlackBerry OS, Nokia's Symbian, Windows Phone platform, and BAD. The survey, which was done by the International Data Corporation (IDC) Worldwide Quarterly Mobile Phone Tracker, depicts global smartphone shipments by the vendor from the fourth quarter of 2009 to the third quarter of 2016. In the first quarter of 2011, RIM shipped 13.8 million smartphones. In 2014, smartphone shipments totaled 1, 230.2 billion units worldwide. In the fourth quarter of 2009, Nokia had a worldwide market share of 38.6 percent (shipped 20.8 million) and was the leading smartphone manufacturer worldwide [30].

Furthermore, the statistic released by website Statista depicts a forecast of the worldwide smartphone shipments by the operating system for 2016 and 2020. For 2020 the shipments of smartphones using Android as the operating system are forecasted to reach around 1.5 billion units. According to data released by IDC, the worldwide smartphone market grew 3.4 percent in the first quarter of 2017 year over year, with 344.3 million shipments [31].

2.4 Smartphones' Impacts

Smartphone has affected almost all area of human life. The outstanding areas, where impacts of a smartphone are notable include business, education, health and social life. Mobile technology plays important role in changing the cultural standards and attitude of smartphone's users. The impacts have two sides positive as well as negative. At one end smartphone are encouraging users to originate their own micro-cultures and share with some activities that may be negatively affected the society and on the other hand, smartphone making connecting all the time for the people is possible [28]. The following sub-sections provide a brief description of smartphones' impacts on society.

2.4.1 Business Sector

Smartphone has formed new dimensions for business and it is also created a new area for mobile application developing companies, Internet services provider and other life's sectors to use the smartphone in order to obtain many competitive advantages [32]. Due to a huge increasing use of smartphones and rapid growth of smartphone and mobile applications, there has been, in the past few years, an extreme growth in the business of broadband and Internet service providers. In a very small duration, a lot of smartphones have been sold that introduced a good opportunity to businesses to invest in mobile application development and enable them to create new business dimensions in the market space. Since it is easy to adjust settings and make personal modifications on a smartphone, hence there are several apps for the smartphone from various mobile companies including Android, iPhone, Microsoft, and Blackberry etc.

The application store is another business sector introduced by smartphones. Different mobile operating system companies have their own mobile applications thus having a different store for mobile applications. The most popular one are Google paly, Apple Store, Windows application store, and Amazon store. These online app stores allow customers to easily download useful mobile applications on a demand basis. Furthermore, these stores provide some free and paid mobile applications. There is no doubt that the widespread of smartphones technology also affected advertising business sector as well. The features of the smartphone have made this sector more productive and without a doubt, it is a more positive influence on a mobile application for business. Mobile application developer, publisher, and service provider are increasing the total number of downloads and revenue by introducing ads as a part of mobile application [28].

2.4.2 Education Sector

The idea and value of education have been superb and noble since the first day in human history and all efforts to enhance the quality of education have been estimated and appreciated throughout. Smartphones introduce another manner for the knowledge seekers to meet their ambitions and expectations. Internet use has become a part of every student's life and a method to search for the information and access requested courses and materials. Currently, the using of mobile phones for Internet surfing has become one of our life routines and a number of the mobile customers surfing the Internet is exceeding Internet users for fixed line [33].

The rapid proliferation of smartphone, fast growth of the Internet communications and high-speed mobile navigation provide an alternative way to present educational materials and services. This will provide an occasion to use the smartphone by students in order to get several educational benefits within unlimited time regardless of their location [28].

Distance education is a learning mechanism that aims to free students from restrictions of time and place and introduces flexible and convenient situation for education. Distance learning gives the students ability to utilize their time and continue their education without affecting their work and family life. The smartphone makes it easier for both students and teachers to cooperate effectively, make teaching easy, help to bring some fun into the classroom, get and share ideas, and more to get the most out learning. Students miss school for health issues or other reasons would be able to attend class or get the learning materials through their smartphones and continue with their work, rather than falling behind due to unexpected conditions [33].

2.4.3 Health Sector

Nowadays, smartphones are the most common devices used for communication and most of the customers are using smartphones for online services. Smartphone's users seem to have a high interest in mobile health apps, especially those concentrate on providing diet and fitness support. According to a study published in the Journal of Medical Internet Research in December 2015 shows that 58 percent of surveyed smartphone customers use their phone to access applications for health and have downloaded at least one mobile health app onto their smartphone [34].

There are a large number of mobile applications built for health care purposes such as the mobile applications that help the users to manage prescriptions, offer alternative treatment options, compare treatment's price and validate prescriptions. In near future, we will see a proliferation of mobile applications, which allow doctors and parents to check a patient/child blood glucose levels at any time during the day. Accordingly to a survey by ITOnline, two-thirds of Americans in the USA use a smartphone to download application related to health information and services in 2015. The survey also shows 79 percent of Americans said they would be willing to use a smart device to manage their health, where a 45 percent wanted tracking of symptoms while 43 percent wanted it to manage their health condition [35].

Smartphones play a key role in the health field. However, today various health apps are available to monitor exercise, diet, and blood pressure. According to data collected by IMS Institute for Healthcare Informatics in September 2015, more than 165,000 mobile health and medical apps available in the Apple iTunes and Android app stores. Among these applications, nearly two thirds are focused on general wellness problems such as fitness, lifestyle, and diet. The remainder of such applications is focused on specific health conditions, medication information and reminders, and women's pregnancy. Use of these apps by a customer is extraordinarily concentrated. They estimated that 12 percent of apps accounted for 90 percent of all downloads; the majority of theses apps are diet and fitness [36].

2.4.4 Human Psychology

The smartphone introduces a new way for users to reduce their stress and pressures in busy work life. Also, it gives users the ability to communicate with their family and friends when they have a free time. The smartphone's users can enhance their social life through communicating with their friends and families while traveling or waiting at a bus station. It enables the users to connect and keep up-to-date with the latest news and development in the social and political arenas.

The smart use of a smartphone will develop the functionality of user's brain rather than using the smartphone for enjoyment purposes only it could be utilized to find useful information, for example, navigate the technology updates, news headlines, and real-time stories from trusted news channels around the world [28].

2.4.5 Social Sector

As a result of the introduction of smartphones, the social life has been partially changed and this sector has faced most of the effects from use of the smartphone. Obviously, in such a situation smartphone plays a positive role in the process of integration with society, particularly, people with special needs and elderly age. Smartphones are capable of enabling this group of people to live more independently [37].

Some examples of smartphone's services such as Global Positioning System (GPS), text to speech and social networking, can support this group of people to easily stay integrated with society. Using these features and many more services, the target group of people can easily interact with their demands, request assistance from others and stay connected to the society [37]. In spite of today's busy world, smartphone had also enabled us to connect with our family and friend all the time. Always communicating to the Internet via a smartphone introduces a magnificent tool for constant connection between individuals, hence a great safety for children when they are at schools or outside the home [28].

2.5 Mobile Apps

With the develop of the 3G service a phone evolves to be a smartphone, and for each new feature it introduces, a new market opportunity is created. A smartphone able to connect to the Internet all the time, this means that developers can promote any service at any given time; airline tickets, personal bank account access, e-mail access, and web surfing.

A mobile application is a software application developed to run on a smartphone, tablet and other electronic mobile devices, and taking in input contextual information [28]. Mobile apps heavily depend on sensed data supplied from context providers, that are sensing such as (light, motion, noise, image sensors) and connectivity like (Bluetooth, GPS, Wi-Fi, 3G, 4G) devices. All those peripheral devices may generate a lot of inputs (e.g., brightness, altitude, temperature, noise level, bandwidth, type of connectivity, and neighboring devices) that vary (even unforeseeable) depending on the environment and/or user actions. By keeping in mind the screen size, the keyboard, internal storage and other hardware specifications, in the mobile handset can be quite restricted. Such online services being discussed earlier can be clearly difficult to implement through the web-based interface.

A web-based service will ask you to open a navigator and surf through the site to get what you need. Since mobile phones are not devices used to explore new services but to solve real life problems and introduce benefits and services to users, these services can be provided as mobile applications that can run at any desired time inside the smartphone's OS.

To continue with the demand of services and applications, some mobile brand names such as Apple and Google, have decided to liberate their developer software, due to this any person able to start developing an application. All manuals and tutorials are available for free for the application software kit which only requires a basic registration. Moreover, different samples of code with different application examples can be freely downloaded to help anyone to be familiar with their programming environment.

2.5.1 What Are the Different Types of Mobile Apps?

If you navigate the list of installed applications on your smartphone, you will find different installed apps with different types. But what are the types of these apps? Are they native, web-based, or hybrid applications? In this section, we describe the different types of mobile app and provide the pros and cons for each of them.

Native Apps

Native apps are developed with a specific programming language to work on a specific mobile platform. For example, Android apps are written in Java, iOS apps are implemented in Objective-C or Swift, and Windows apps using Visual Basic or C++. Native apps are able to use all platform-specific libraries and operating system APIs (application programming interface), in order to get the advantage of all the features that offered by a modern smartphone [27].

For instance, if the user has granted the necessary permissions, the app has a direct access to the platform-specific hardware and software features such as camera, Bluetooth, sensors, and GPS. Hence, developers are able to create apps that utilize the system capabilities such as the GPU and CPU in order to create powerful apps. Native apps have excellent performance since they are optimized for mobile platforms. Generally, native apps have good look and feel and they have a full support for all touch screen gestures. As well as the native app can be distributed easily on app stores.

The development work is a difficult task due to a large number of resources that can be found. The app code must be written particularly for every targeted mobile platform, the same code will be rewritten with little that can be shared. The logic may be the same, but on the other hand, the programming language, APIs, and the development approach are different. This approach can be sometimes long if the application is complex. Moreover, the approval process can be quite long.

Web Apps

Mobile web apps are really websites that can be run by the device's web browser. This category refers to a standard web application developed using traditional web technologies such as HTML and JavaScript, particularly with HTML5, CSS3, and JavaScript. Users can access web apps as they would access any web page: they navigate to a web page's URL and then have the choice of "installing" them on their home screen by using a bookmark option.

Web apps became really popular when HTML5 was introduced and developers observed that they are able to utilize native-like functionality in the browser. HTML5 enables developers to create mobile websites with animated and interactive components. They can include audio or video objects and use positioning utility. Nowadays, it is easy to develop mobile web apps by using of HTML5, CSS3, and JavaScript.

There are two main ways to make the web application suitable for smartphone devices. The first way uses responsive design, which dynamically rearranges page layout. This is supported with new features of CSS3. The second way checks information received from user HTTP request to decide whether the customer uses a mobile web browser. If so, redirect the customer to the website designed for smartphone devices.

Even the web apps have a full access to some device resources which are provided by the web browser - in most cases, it is possible to access GPS, camera or accelerometer, but cannot access near field communication (NFC) or Bluetooth. In comparison with the native and hybrid apps, web apps tend to be slower than because they need to download all the web objects and elements that are displayed on the screen.

In comparison with the native and hybrid apps, web apps are faster and cheaper to implement. They are mobile platform independent. Furthermore, no submission for mobile web app store is required because web applications are not installed from application stores and developers can easily and quickly update them. Mobile web apps strictly depend on the mobile browser; hence, the mobile web apps can work and behave differently since the standards of HTML5, CSS3, and JavaScript are not fully supported by all mobile browsers. This can have a major impact on the mobile web app. To overcome this challenge, the application testers should consider all the available browsers.

Hybrid Apps

This is previous technologies mixture. Hybrid apps are built with a combination of different web technologies such as HTML, CSS, and JavaScript. Actually, they are built in a similar way as websites. The hybrid apps are hosted inside a native application that utilizes a mobile platform's web view. This enables them to access device features such as the camera, accelerometer, contacts, and more. In comparison, hybrid frameworks provide better and wider access to device features than web applications.

Further, hybrid mobile apps can contain native UI components in situations where necessary. But the quality of UI, security, and performance are often worse as compared to the native applications. Hybrid apps are often built to be platform-independent, so their main characteristic and advantage are portability. There are several leading approaches how to build hybrid applications. The most common used cross-platform frameworks utilize web technologies.

After the web part has been built, developers can compile this code base to the different native formats: Android, iOS, Windows Phone, or BlackBerry. The HTML content or elements of hybrid apps can be hosted on a server. As a result of this, the developers able to perform small updates easily without updating the whole app through the app store submission process.

Hybrid apps are a great technique to append the web elements and package them up to be published later in an app store. You can publish hybrid apps in any of the major app stores: Google Play, Apple App Store, Microsoft Windows Store, Amazon App store and even BlackBerry World. Furthermore, hybrid development is an easy task; there is one code base for every targeted mobile platform. For hybrid apps, matching the design guidelines of the various mobile platforms is not easy as well as the approval process may be long.

2.6 App Stores

App Store is a digital distribution platform most smartphone companies have. It is considered as the core of the mobile world where apps can be distributed, reviewed and downloaded. App stores allow users to browse and download applications that are developed and enable third-party such as free programmers and software companies to develop applications. In additions, they present a variety of services and benefits; sometimes they add extra value to their devices. With the app stores, mobile devices such as smartphones and tablet computers would be more intelligent, attractive and functional.

Nowadays, the most widely accessed and biggest mobile app stores are Google and Apple which they contain more than 5 million apps [38]. Meanwhile, huge numbers of mobile apps are also distributed and downloaded for mobile devices, particularly from Apple's App Store for iOS devices and Google Play for Android devices. Google Play, the largest digital distribution platform for mobile apps, which is operated by Google, provided a collection of 2.8 million apps as of March 2017 [38]. More than 65 billion apps had been downloaded from its app store from August 2010 to May 2016 [39]. On the other hand, Apple store is catching up with 2.2 million apps in March 2017 [38]. Apple announced that more than 140 billion apps had been downloaded from its App Store from July 2008 to September 2016 [40].

In addition to the two big app stores from Apple and Google, there are other app stores maintained and operated by device manufacturers and network operators. The following list is not complete but shows some other app stores for the different mobile platforms:

- Windows Store
- Windows Phone Store (formerly Windows Phone Marketplace)
- Amazon Appstore
- Blackberry World
- Nokia Store
- Mac App Store
- Samsung Apps
- F-Droid

However, before you publish your apps to one of the stores and distribute them to users, you need to get knowledge about the store's review and publishing process. Your app requires to meets the review and publishing guidelines of the different companies and app stores; otherwise, your app will be rejected. When you become familiar with how the process works, you will help your team while preparing and releasing your app.

Chapter 3

Mobile Operating Systems

Good knowledge of smartphones' platforms and applications development environment is essential for every mobile application tester. In this chapters, we present Android, the most common and used mobile operating system, and then we briefly describe iOS, Windows and Windows Phone, and finally provide you with a short overview of the current state of mobile app stores.

3.1 Android

Android is one of the most popular and widely spread mobile operating system (OS) for smartphones as of 2016. It is also the only well-known operating system that dominated the smartphone's market with a share of 86.8 percent during the last year according to IDC (2016), beating both Apple's iOS and Microsoft's Windows Phone [31]. Android Inc. was established by Andy Rubin, Rich Miner, Nick Sears and Chris White in 2003, but was later acquired by Google Inc. The initial purpose of the company was to create an advanced operating system for digital cameras. Though, when it was perceived that the market for the intended devices was not huge enough, the company diverted its plans toward releasing a smartphone operating system that would compete with Symbian and Microsoft Windows Mobile [41][42].

The development was then diverted towards smartphones, and in 2007 the Open Handset Alliance was established, with companies such as Google, Sony and device industries such as HTC, and declared open standard development for mobile devices. Android is owned and operated by the Open Handset Alliance-industry consortium for creating hardware, software and telecommunication open standards for mobile devices. This consortium is led by Google [43]. The Android operating system was developed as their first product, a mobile device platform built on the Linux kernel. In 2008 the first commercially smartphone (HTC Dream) was created using Android as the running operating system [42].

Furthermore, Android is an open source OS where every developer can download the source code and compile their own version, it is strictly based on the Linux kernel and

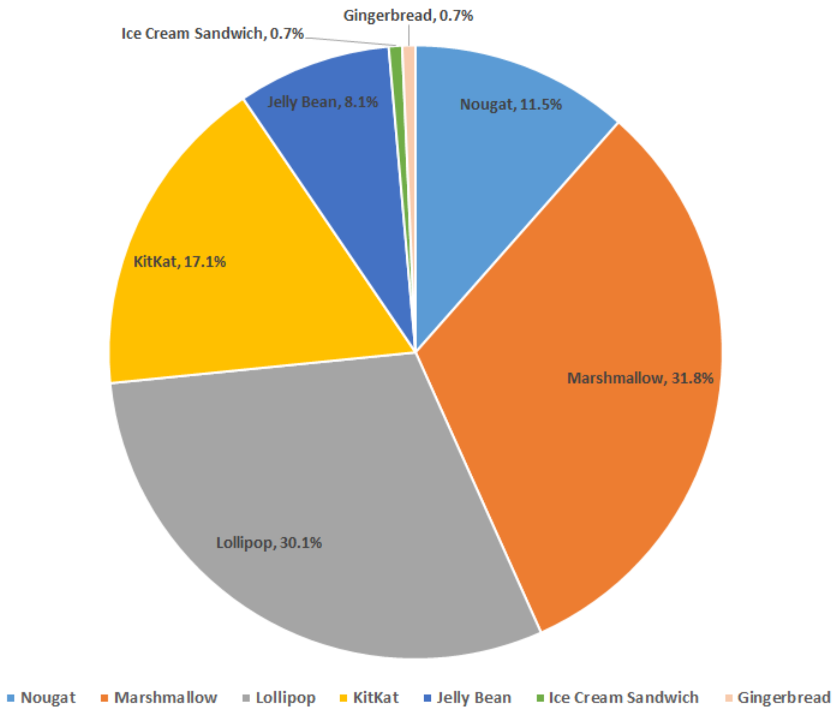


Figure 3.1: The distribution of the Android versions [44].

designed primarily to support touchscreen mobile devices such as smartphones and tablets [42].

The Android's open nature has encouraged most manufacturers and cellular carriers to take advantage of that and customize the operating system to meet the requirements of their hardware or bring Android to devices which in origin shipped with other operating systems. Due to its popularity, many testing tools and frameworks such as Robotium and Espresso target Android.

There are a number of different versions of Android operating systems. Each version named alphabetical order after a codename that represents a piece of dessert, for example, version 7.1 "Nougat". Figure 3.1 illustrates the distribution of each version as of July 2017, where Marshmallow is sitting as the most widely used Android version, running on 31.8 percent of all Android devices, while Nougat runs on 11.5 percent of devices, which was released on October 4, 2016 [44].

3.1.1 Android Software Development Kit

Android developers can develop applications by accessing the Android software development kit (SDK), which is basically a set of tools that are helpful to develop Android applications. Android applications are packaged into APK files. An APK file contains all the assets, compiled source code (such as .dex files), resources, certificates, and manifest file which are needed for distribution and installation of mobile apps on a device [42].

To write test cases for Android applications, the Android SDK includes special tools and provides the developer with capabilities to write automated tests using the UI Automator framework, and Monkeyrunner which enables mobile testers to write programs that control an Android device or emulator, generate pseudo-random events from outside of Android code, and take screenshots for application's GUI [42][45].

Android software development kit enables developers to build applications, which extend the device's functions. Often, the Java programming language is used to create applications since it has complete access to the Android APIs. Java may be integrated with C/C++, together with an option of non-default runtimes that allow better C++ support.

3.1.2 Android Software Stack

The Android architecture is often called the Android software stack which is roughly categorized into five components as shown in Figure 3.2 [46][47].

Linux kernel. The heart of the Android platform is the Linux kernel. Linux Kernel exists at the root of Android architecture. However, the Android Runtime (ART) depends on the Linux kernel for underlying functionalities such as threading and low-level memory management. It is responsible for device drivers and enables resource access such as interacting with the display card, camera, Bluetooth, audio devices. Linux kernel allows Android to utilize the advantage of key security features and allows device industries to create hardware drivers for a well-known kernel.

Hardware abstraction layer (HAL). The hardware abstraction layer supports standard interfaces that present device hardware functionalities to the Java API framework. The HAL provides multiple library modules, each of which implements an interface for a particular type of hardware components, such as the Wi-Fi or Bluetooth module. When a framework API performs a call to interact with the hardware component, the Android loads the library module for that device hardware.

Android runtime (ART). In smartphones which are running Android version 5.0 (API level 21) or higher, the mobile app runs in its own process and with its own instance of the Android runtime. ART is built to run numerous virtual machines on low-memory devices by executing DEX files, a bytecode format built mainly for Android that is optimized for minimal memory footprint. ART converts the translation of the application's byte-code into native instructions that are executed later by the runtime environment of the device.

Android runtime includes the following main features:

- Ahead-of-time (AOT) and just-in-time (JIT) compilation
- Improved garbage collection (GC)
- Better development and debugging environment, including a sampling profiler, improvements for diagnostic detail in exceptions and crash reports, and the capability to set watch-points to control certain fields

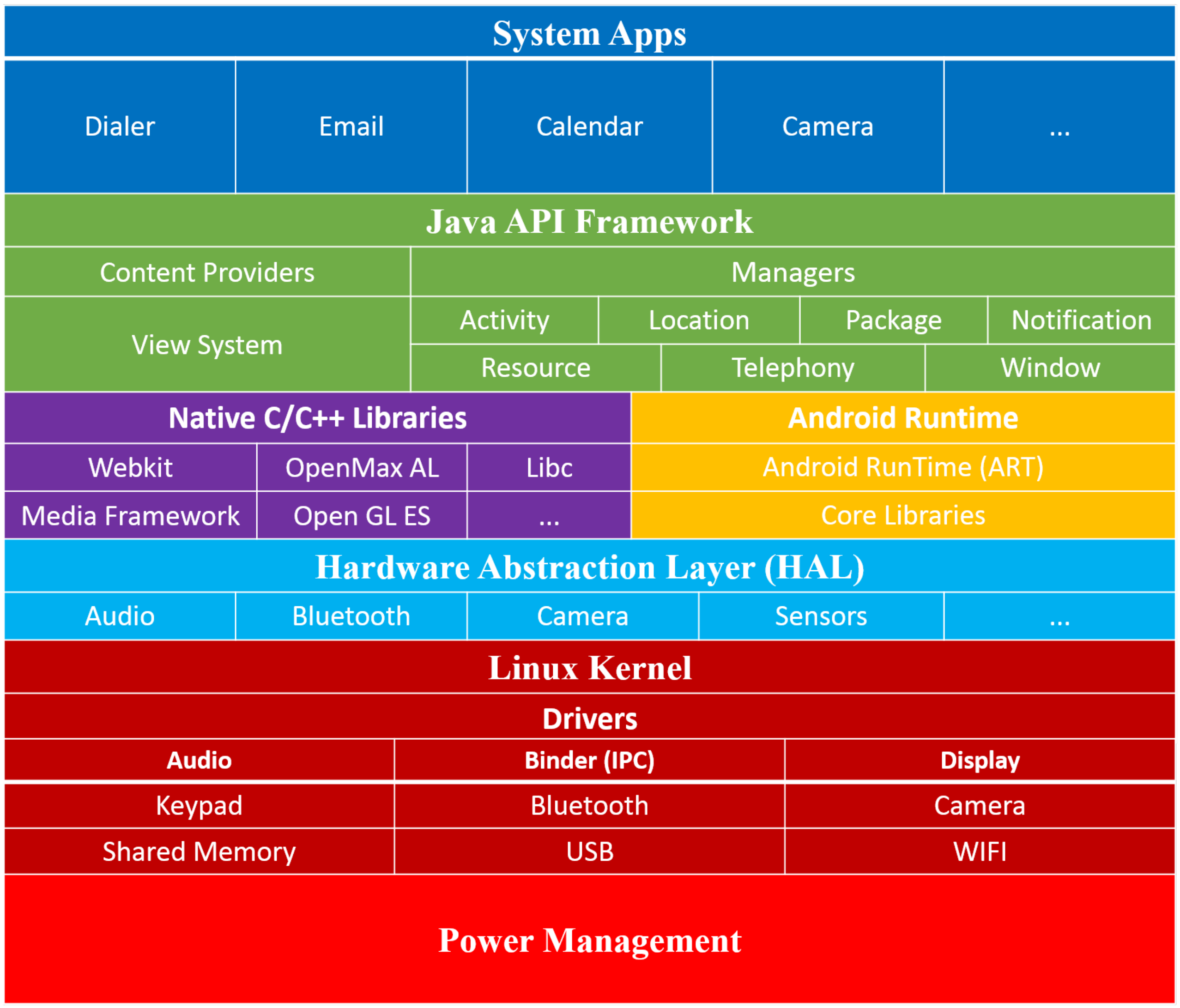


Figure 3.2: Android software stack [46].

Before Android version 5.0 (API level 21), Dalvik was the Android runtime, it is used to run Java-based applications. Dalvik Virtual Machine (DVM) is similar to Java Virtual Machine (JVM) but it is correctly adjusted for mobile devices with low memory consumption and high performance. In addition, Android provides a group of core runtime libraries that supports most of the functionality of Java programming language.

Native C/C++ libraries. On the top of HAL, there are native libraries. For Android's developers, to be able to operate properly with its core features, the libraries layer exists to support functionality such as 3D rendering using SGL (Scene Graph Library), connecting to databases using SQLite, and drawing and manipulating 2D and 3D graphics in mobile app by accessing OpenGL ES.

Android versions were supplied with DVM, which is utilized to interpret the bytecode that is produced from compiling Java-code. When Lollipop released, DVM was replaced with ART as runtime environment. Many core Android system components and services, such as ART and HAL, are developed from native code that needs native libraries written in C and C++. The Android platform introduces Java framework APIs to reveal the functionality of some of these native libraries to apps. In case the mobile app is developed using C or C++ code, the Android Native Development Kit (NDK) can be used to access some of the native platform libraries directly from app's native code.

Java API framework. On the top of native libraries and Android runtime, there is a Java API framework. Java API framework supports the entire feature-set of the Android OS which built using Java language. These APIs provide the building blocks which developer needs for android application development by simplifying the reuse of core, modular system components, and services, which consist of the following:

- A View System, with rich and extensible form, a developer can use to build an app's UI, including lists, grids, text views, edit text, buttons, and even an embeddable web browser.
- A Resource Manager, enabling access to non-code resources such as localized strings, styles, drawable files, and layout files.
- A Notification Manager that lets all apps to show custom alerts in the status bar.
- An Activity Manager, allowing management for the apps lifecycle and introduces a common navigation back stack.
- Content Providers that gives apps ability to connect to data from other apps, such as the contacts app, or to share their own data.

System apps. The top layer is the applications layer, which handles a set of core apps for email, SMS messaging, calendars, internet surfing, contacts, and more. Apps installed on the mobile device have no distinctive status among the apps the user can download. Hence, a third-party app provides important services and can be the user's default web browser, contact, SMS messenger, or even the default soft keypad. Developers can easily utilize and access the system apps function capabilities from their own app.

3.1.3 Activities

Activities are one of the primary building blocks of apps on the Android platform. They act as the entry point for a user's interaction with an app and are also act as a mean for user navigation within an app or between apps. Activities are used to place and display graphical elements for the user to interact with. Android applications usually consist of multiple or at least one activity that is bound to each other, and the activities can switch between each other to display various fragments of the application or execute different actions. Even though activities are often displayed in a full-screen window, they can also be displayed in other modes: as floating windows or embedded inside of another activity [48].

By using an intent object, activities can invoke other activities, which is like a glue between the activities in the application. Intents are basically messaging object used to request an action from another app component. Intent is also used to start a new instance of an activity, and enable the user to send extra data to the started activity, a bundle can be invoked, which acts as an intermediate storage when transferring data between activities. Activities in the Android app are managed using the stack. When a new activity is started, it is pushed to the stack and placed on the top with running state, the previous activity always placed below it in the stack, and will not come to the front again until the new activity exit far. An activity has basically four states [49]:

- Running state: when an activity is in the front of the screen (at the top of the stack).
- Paused state: when an activity has no focus but is still visible (that is, a new not full screen or transparent activity has focused on top of previous activity).
- Stopped state: when an activity is completely covered by another activity, however, it is not visible to the user so its window is hidden and it might be killed by the system when memory is full.
- Finished state: when an activity is paused or stopped, the system can erase it from memory by either asking it to finish, or simply terminate it.

Figure 3.3 shows all important types of states available for activities.

3.1.4 Publishing

Google Play[1] is the main digital platform for application distribution, but users can also download applications from other stores like Samsung Apps or Amazon Store. Applications on Android are easily downloaded and managed through Google Play. According to Statista (March 2017) [38], Android is the leading app store for a number of applications available on their digital distribution platform, with 2.8 million applications as shown in Figure 3.4.

[1]https://play.google.com/store

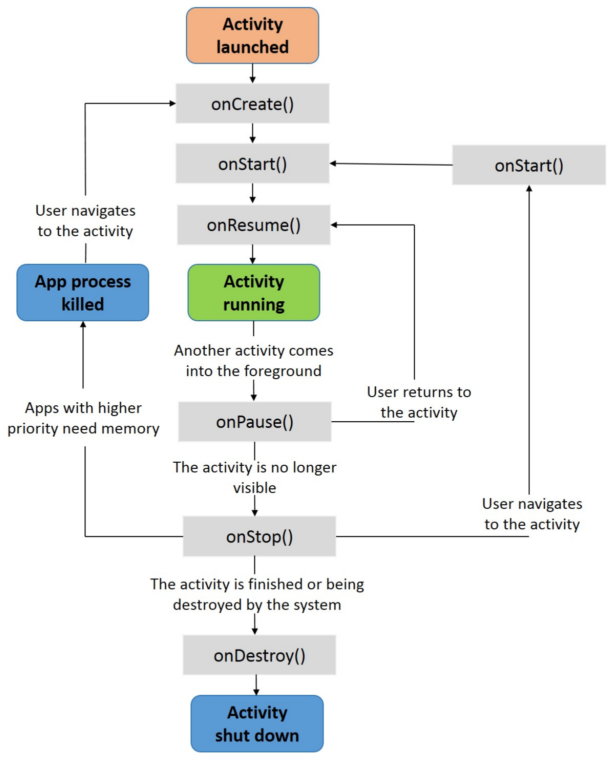

Activity launched
onCreate()
onStart()
onStart()
onResume()
User navigates to the activity
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finished or being destroyed by the system
onDestroy()
Activity shut down

Figure 3.3: A simplified illustration of the activity lifecycle [48].

Developers need to access the Google Play console and to register using valid Gmail ID for applications publishing on Google Play. It is recommended to get knowledge about the Google Play launch checklist[2] before releasing an application to Google Play. This checklist helps developers to prepare the application for publishing. Moreover, the checklist helps developers to understand the publishing process and get ready for a successful product launch on Google Play, it provides information about Google Play policies, localization, recommendations for testing, design guidelines, and tips for requirements of the app. The whole checklist can be found at the Google Play developer site [50][51].

3.2 iOS

iOS is a mobile operating system created and owned by Apple Inc. It is the UNIX-like operating system based on Darwin (BSD) and OS X, further, it is categorized as closed source code. iOS was originally developed to target iPhones, but now iOS works on iPad, Apple Watch or Apple TV. It is the second most popular mobile operating system framework by sales in the world, after Android.

Most users of iOS (95 percent) have the iOS version 9 or the latest 10 [52]. Consequently, testing on all available devices is not difficult as compared to Android. On the other hand,

[2]https://developer.android.com/distribute/best-practices/launch/launch-checklist.html

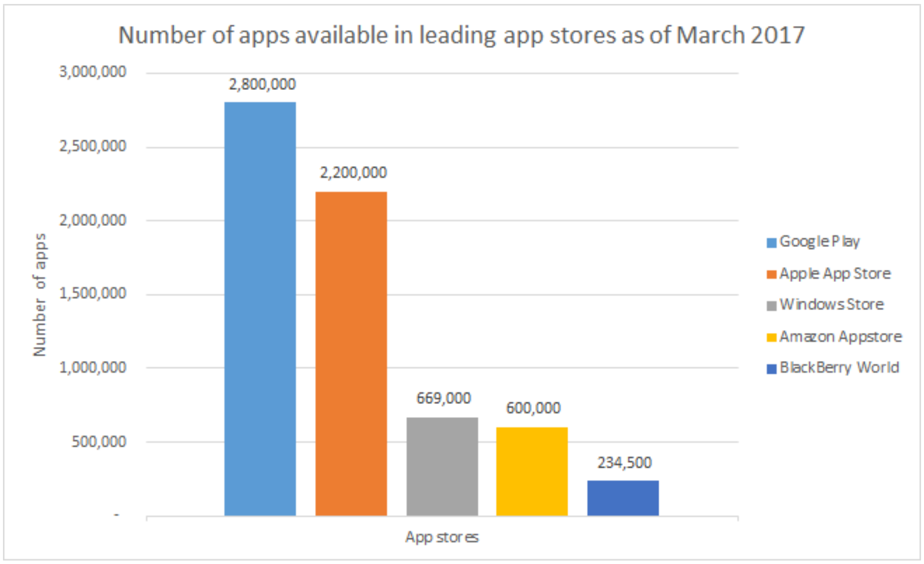


Figure 3.4: The amount of applications available in leading app stores [38].

the testing costs for Apple devices would be high due to the purchasing of devices are too expensive.

3.2.1 Publishing

App Store[3] is the only one digital platform for an application distribution. According to Statista website (see Figure 3.4), Apple users were able to choose between 2.2 million apps [38]. To release the application on the App Store, Apple ID and iOS developer account are required. There are two types of accounts offered by Apple – individual and enterprise.

Applications' submission to the App Store has to pass the process that based on a group of technical, content, and design standards, with the special constraint on user privacy and functionality. The application should fulfill the App Store review guidelines[4] and iOS human interface instructions. The application on the App Store should be useful and meets the user's expectations with high performance and quality. During application approval process, the application is tested using the two types of testing: automatic and manual [52].

3.3 Windows and Windows Phone

Windows Phone is a mobile operating systems developed and owned by Microsoft. It is especially for smartphones. There are two types of applications projects developed by Microsoft - Windows applications for PCs or tablets and Windows Phone applications

[3]https://itunes.apple.com/us/genre/ios/id36?mt=8
[4]https://developer.apple.com/app-store/review/

for smartphones. Particularly, a lot of APIs, tools, and design principles are shared by these two projects. Hence, they were partially integrated to the Windows Universal Applications, which allows sharing of code between Windows Store and Windows Phone applications projects [53]. Nowadays, the future of Windows phones is unclear and far from guaranteed, and the Catrobat's developers are wondering whether Windows has a future on smartphones at all, however, they did much effort to develop a version for Windows OS.

3.3.1 Publishing

Windows Store and Windows Phone Store[5] are exclusive application digital platforms distribution, which will be also integrated into Universal Windows Store in Windows 10. A single developer's account is required by Microsoft to the both stores, as well as developer's account can be individual or company.

Microsoft applies slightly different approach for approval process than the others. During the approval process, the application content compliance is checked manually but other tests are automated (security, technical compliance) [53]. Uploading process for the application on Windows (Phone) Store is very similar to uploading process on Google Play which is described in section 3.1.4. As well as the approval process is fast as compared with the others.

[5]www.windowsphone.com/en-us/store

Chapter 4

Key Challenges in Mobile Application Testing

Unlike the PC environment, the mobile environment consists of endless or plethora of devices combinations, OS models and screen sizes with various hardware and software configurations and communication. Hundreds of different smartphones are on the market, designed by different vendors, and with different software features and hardware devices. This makes the testing of mobile web and mobile apps a much more difficult task than the testing of websites and desktop applications; testing on many different devices and versions can become very costly and time-consuming.

When automated testing of mobile application is performed precisely and perfectly, it can potentially minimize test development and run time dramatically and accelerate the time to publish the mobile app into the market. There are, of course, other topics that make mobile testing a special and challenging task. This chapter contains the key challenges that affect the success of automation testing based on [27][54][55] and my personal experiences.

4.1 The Potential Customer

To meet customers' requirements, it is really essential to collect information about the potential target customers and their expectations and demands. If the mobile company releases an app without any kind of background of the target users, the app will most likely not be installed or it will get a really low rate. This causes fewer downloads, the app will die, and customers may try to download an app from the company's competitor.

Without a doubt, if the developers want to handle such challenge, they need to collect as much information about their actual and potential end users as possible. This means that during the development and testing phases, developers need to integrate the details of the target group, such as age, gender, and geographical background. However, Figure 4.1 illustrates the information that mobile tester needs to collect.

Moreover, to increase customer's satisfaction level, it is recommended to invite customers

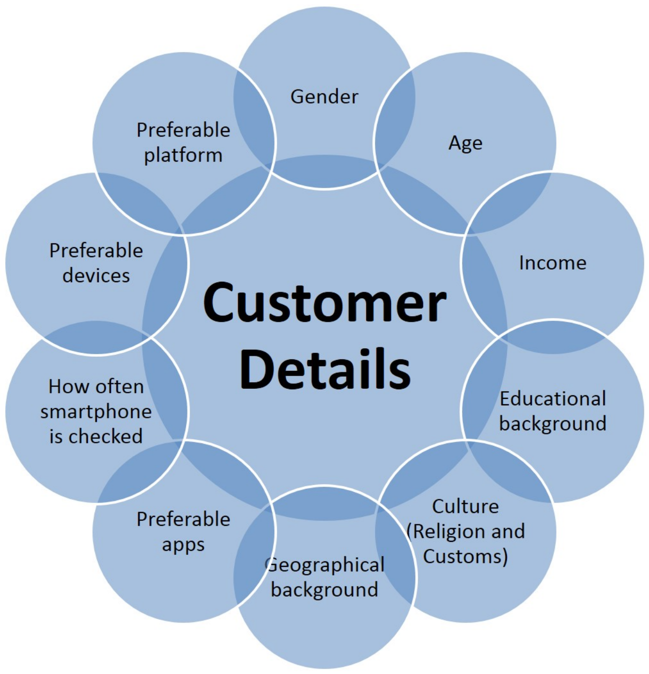


Figure 4.1: Target group details.

to usability tests, interview customers and get their feedback about your product, invite them to be beta testers and check and learn for the app store reviews and useful comments.

4.2 Multiple Platforms and Fragmentation

In the previous chapters, we introduced the different mobile vendors and mobile platforms. For some mobile platforms, there is more than one mobile device manufacturer, actually, numerous mobile versions running simultaneously in the market and this lead to inevitable fragmentations. Fragmentation is a great issue in the mobile world and particularly in the Android world, and it is considered as one of the biggest challenges for the Android developers since there is a wide range of end devices and operating system versions [54]. Due to such issue, Android apps frequently suffer from cross-platform and cross-version incompatibilities, which makes manual testing of such apps expensive, and therefore, particularly worth automating.

According to a study conducted by OpenSignal [56], in August 2015 there were more than 24,093 different Android devices are available in the market, all of which differed in type, shape, screen resolution, and specific configurations and with vastly different performance levels and screen sizes. Practically, it is not necessary and possible to test on all of those devices. Figure 4.2 shows the distinct Android devices were increased dramatically between 2012 to 2015. As can be seen, the distinct Android devices in 2012 were 3,997. Then there was a great increase in 2013 to reach 11,868, the number of distinct devices increased from 11,868 to 18,796 in 2014, and finally, in 2015, it reached 24,093.

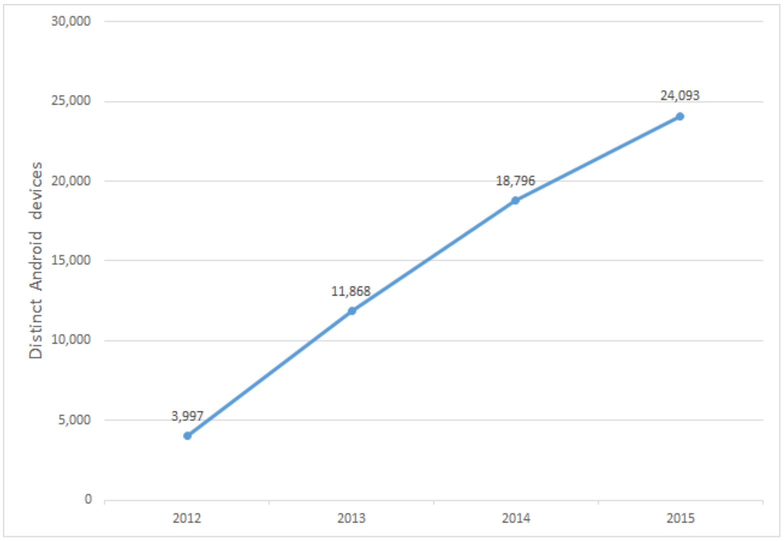


Figure 4.2: Android fragmentation.

This issue is not just limited to Android; other mobile platforms such as iOS, Windows Phone, and BlackBerry are also influenced by fragmentation. The possible combinations of hardware and software on those platforms can also be considered as a problem [55].

4.3 Sensors and Interfaces

Nowadays, the smartphone is incredible little devices, has a variety of sensors and interfaces which already installed on a user's phone, these sensors can be accessed by the installed apps to introduce useful and exciting features to users. They also aim to gather data for various user purposes, often in conjunction with a mobile app. The actual sensors and interfaces used to rely on the app's functionality and objectives. While mobile applications are running on different devices, they may react differently because of variations in the peripheral devices or operating system versions. For instance, sensors are typically calibrated differently, so that two or more (different) smartphones running the same application, on the same platform, may produce different outputs. Therefore, it is the responsibility of mobile tester to make sure that all the implemented sensors and interfaces are running correctly. It is also important to verify that malfunctioning and failing sensors do not negatively affect the app.

4.3.1 Ambient Light Sensor

A phone's light sensor measures how much light is available in the current environment (i.e., a brightness of the ambient light). The phone's software uses this data to automatically adjust the screen's brightness and automatically display accordingly, it can prolong the device's battery life, which is a key benefit in smartphone applications. However, this sensor works well in all types of light sources from natural sunlight to fluorescent and incandescent lamps. Furthermore, when ambient light is plentiful, the display's brightness is pumped up, and when it is dark, the screen's brightness is dimmed down.

4.3.2 Acceleration Sensor

The acceleration sensor measures how the changes in the device's orientation (tilt and vibration), and detects the proper acceleration that the handset is experiencing relative to free-fall. The most common use for this sensor is when the device's mode is changed between portrait and landscape. Usually, this sensor is accessed by the app if the developers implemented a portrait and landscape features. Frequently, during the testing phase, the mobile tester should change the orientation of the device. Probably, UI glitches are detected since the UI elements could be moved and their positions are changed. Also, the app might be crashed, for instance, when data is retrieved from the backend while a UI layout is being refreshed.

4.3.3 Proximity Sensor

Another helpful sensor is the proximity sensor, which is consist of an infrared LED and an IR light detector, it detects the presence of nearby objects without any physical contact, such as to a face or surface. For a good reason, it can be used to automatically turn off the screen display when a user places the handset up to her ear to avoid unintended input. More specifically, the sensor informs the system that user most probably in a call and that the screen has to be switched off. This prevents a user from unintentionally tapping or touching buttons on an active screen. It is also helpful for saving a device's battery life.

4.3.4 Magnetic Sensor

Another sensor that most smartphones now have is the magnetic sensor. It is able to measure and detect the strength and orientation of magnetic fields around the smartphone. This sensor is mostly used by compass applications to point at the planet's the North Pole, and also for navigation purposes. With the help of this sensor, the smartphone has the ability to determine the orientation in which its facing: west, east, north, or south. If the app uses the magnetic sensor, it is recommended to test it in different environments. For instance, if compass app is used in a building where it is surrounded by lots of minerals, the magnetic sensor may give incorrect information or data, which can lead to inverse side effects in the app.

4.3.5 Gyroscope Sensor

The gyroscope sensor is used to either detect the current orientation or changes in the orientation of the device. The main difference between an accelerometer and gyroscope sensor is that an accelerometer measures the linear acceleration of a device, while a gyroscope exactly tracks the device's rotation and twist. This means that the sensor is able to detect 360-degree motion, with greater precision. With the help of accelerometer and gyroscope sensors, the device is able to support the six directions: left and right, up and down, forward and backward. However, mobile tester should keep the six axes in his mind when testing the target app.

4.3.6 Temperature, Pressure, and Humidity Sensors

These three sensors are not embedded in every smartphone yet, but they will be soon. The three sensors can be used to gather more information about the user's current location to provide apps with useful information such as the temperature sensor measuring the ambient temperature, pressure sensor can measure atmospheric pressure and this measured data can be used to determine how high the smartphone is above sea level, and humidity sensor measuring the current and ambient humidity. These sensors are accessed, for example, by outdoor or weather apps. Further, a mobile tester should test these sensors in different environments with different temperatures, pressures, altitudes, and humidities.

4.3.7 Location Sensor

By using this sensor, apps can detect the smartphone's latitude and longitude as well as a street address. Location sensor or GPS is used in many various types of apps such as map apps, camera apps, and social media apps. In such apps, you can use it to share your location with others, and let your friends know your current position and where you are. Certainly, the functionality of GPS should be tested in different locations, such as in the countryside or in downtown sprawl with lots of big buildings and crowded area.

4.3.8 Touchless Sensor

The touchless sensor is not included into every smartphone. In most cases, this sensor is embedded into the front of the device to receive touchless gestures from a finger or a hand. For instance, the user can swipe between menu elements in her device simply by waving her hand over the screen. If your app implements touchless gestures, a mobile tester should make sure every gesture correctly works at various angles and orientations.

4.3.9 Fingerprint Sensor

It is built into a number of smartphones; this sensor is mostly used by apps that require security information. However, this sensor is convenient to use since it is able to read fingerprint data without finger swiping. Often, fingerprint scanners are most used as an extra layer of security to replace a lock screen password.

4.4 Touchscreen

The most important interface in smartphones is the touchscreen. Users can interact with a smartphone by touching the screen with a special stylus and/or one or more fingers to generate gestures that the phone is able to translate them into instructions. The user can react to what is displayed on the screen and to control how it is displayed; for instance, zooming to increase the size of the text.

Basically, there are two types of touchscreen technologies available. The first is the resistive screen that is made out of different layers and responds to pressure. This type of touchscreen is usually created for use with a stylus. Whenever a customer wants to sign for a package delivery, she probably signs on a resistive screen. The most common disadvantage of this technology is that it does not support multi-touch gestures [27][55].

That is why the necessity for the second technology is increased. The capacitive touchscreen is a technology which nowadays available on a smartphone. It is responding to touch instead of pressure and supporting multi-touch gestures.

Capacitive screens contain an insulator, it is a glassy thing coated with a transparent conductor like indium tin oxide. Touching the capacitive screen by the human's finger creates a distortion of the screen's electrostatic field. Then, the generated distortion is translated to data that the smartphone can read and interpret [27].

The following are the most common gestures on a capacitive touchscreen:
Touch: Touch the screen using a fingertip.
Swipe: Move fingertip over the display screen.
Tap: Briefly touch the screen using a fingertip.
Double tap: Briefly touch the screen two times using a fingertip.
Drag: Continuously move a finger over the screen.
Multi-touch: Use two or more fingers on the screen at the same time.
Pinch open: Touch the screen using two fingers and move them apart.
Pinch close: Touch the screen using two fingers and bring them closer together.
Rotate: Touch the screen using two fingers and then rotate them.

The diversity of possible touch gestures introduces a special challenge while testing a mobile app. The mobile tester should put all the possible touch gestures in his consideration and implement them while testing.

4.5 Microphones

Another way for users to interact with an app is by using their voice or sound. The microphone is for transmitting user's sound electronically through the call. Other microphone would be used for better audio recording. Usually, smartphones have more than one microphone included. Actually, there are up to three microphones which are located at different positions: front, back, and bottom. However, these three microphones guarantee high quality for voice recording from all possible directions and regardless of the smartphone's position. When testing voice inputs via the microphone, the mobile tester should test voice input indoors with normal and noisy situations, and test the app outdoors with a lot of noises. And finally, increase or decrease the sound level by using the volume up and down buttons and at the same time check the behavior of the app.

4.6 Camera

Today, most of the smartphones already have two cameras: rear and front. The rear camera is utilized to take high-resolution photos, and the front camera is usually utilized for video calling with a lower resolution. Most smartphones only have a menu option to launch a camera application program and an on-screen button to activate the shutter. Cameras are commonly used in several of mobile apps for capturing photographs and often recording video. Also, smartphones can use their touchscreen to point their camera to focus on a certain object in the area of view.

Some apps access the camera to scan information using OCR (optical character recognition) or other types of shapes. If the app uses the camera, test it with a different of mobile devices that match the target customer needs. Every smartphone has a distinct camera with a distinct lens, resolution, and flash. The different resolutions of the camera have an effect on the image size. The mobile tester should cover the functionality of each camera with different resolutions in order to check if the app is able to handle both small and large images. Moreover, the performance of the app should be checked while the camera mode is activated [27].

4.7 Keyboard Layout

A keyboard enables users to enter text. It is not a GUI aspect; rather, it is a system aspect. Some industries offer a keyboard framework that lets users create on-screen input methods such as virtual keyboards and sometimes these keyboards apps are pre-installed and considered as system apps. Such apps are likely to have an effect on an app. On Android phones and tablets, customers are able to replace the pre-installed keyboard app with special keyboards that support a totally different style of typing and keyboard layouts. There are traditional tap keyboards (like QWERTY), keyboards with various layouts (diverse characters per key), keyboards support the functionality of tap-slider, and keyboards that support a swiping technique to insert text.

Replacing the preinstalled keyboard can have an impact on a mobile app. For instance, the keyboard may be much bigger than the default one and therefore hide UI elements that are necessary for users to communicate with. Further, it is also possible the keyboard settings are lost within the device, and the app might cause freezes or crashes. Finally, mobile testers should keep in their mind that if the app interacts with one of the pre-installed system apps, the users are able to replace them with other special keyboards [24].

4.8 Internationalization and Localization Processes

Another challenge that needs to be considered during the mobile development process is the internationalization and localization of mobile apps. Software internationalization and localization are important steps in deploying software to different locales of the world [10]. However, localization is not just about translating all strings and customizing of the software user interface but also making sure the product matches to a local user experience. It aimed at developing an accessible, usable and culturally suitable product for a specific locale [6].

The difference between internationalization and localization is subtle but essential. Internationalization refers to the process of designing and developing a system that supports different languages and regions, while localization is the process of translating an application's resources into versions for each specific locale that the application supports [6][16]. Furthermore, it means adding special features for the specific region. Practically, internationalization activities are performed once per product. Whereas localization activities are performed once for every combination of product and locale. The internationalization and localization activities are complementary and must be integrated to lead the purpose of a system that works globally.

The quality of international software is completely dependent on the software's localization level; therefore, localization testing is an essential part of quality assurance of international software and thus has turned into a major type of international software testing. Late changes because of translation issues can delay the release date of the app or negatively impact the app's design. It is really important to test both I18n and L10n issues if the app is developed to be used in different countries. The mobile tester must ensure that the different languages will not break the UI elements or have an effect on the app's usability [6].

Many languages have their own character set, and one word can have a very different width and height, so more space is allocated than necessary for English to accommodate most other languages. Have a look at the example word *settings*. When we compare the word with translations from Germany, France, and Arabic, we will observe lots of difference in characters, reading direction, width, and height:
Settings (English)
Einstellungen (German)
paramètres (French)
إعدادات (Arabic)

Both the German word (Einstellungen) and the French word (paramètres) are much longer than the English word (Settings), while the Arabic word (إعدادات) with a different text direction (right-to-left). Moreover, Asian languages can be considered as "short" languages, while German and Swedish are "long" languages. All translations can lead to a UI glitch, string truncation or even break in app design rules. When testing an app in various languages, check that every UI element is flexible enough to accommodate several language fonts and strings lengths and that every screen has consistent look-and-feel [6].

The same process should be applied to the following issues that are used within apps [6][57]:

- Locale and culture awareness (formats for numbers, dates, times, addresses, and units of measurement)
- Layout direction (left-to-right script vs. right-to-left)
- Currency format
- Text directionality (left-to-right script vs. right-to-left)
- Language character coding sets for textual display
- Name and titles
- Sorting and indexing rules
- Case conversion

4.9 Multiple Mobile Browsers

Mobile web browsers are designed to display web elements such that they can be accessed on smaller screens. A tester needs to test the mobile web app in a mobile web browser. However, there is more than one web browser for mobile available in several versions of the different mobile OS [24][55]. In addition, the browsers use different layout engines such as the following:

- Blink
- Gecko
- Presto
- Trident
- WebKit

The layout engine, browser configurations, and browser version play a great role how the mobile web apps may look and behave. This is an especial case for the different browser layout engines. Every browser layout engine manages languages like HTML, CSS, and JavaScript in a different way. Not every browser has the completed implementation for all feature set or the latest version of the different languages (HTML, CSS, JavaScript), which can, of course, cause variation in output [27].

To make sure that mobile web app runs properly on different browsers, the app should be tested on different mobile OS such as Android, iOS, or Windows Phone together with different browser versions. However, testing mobile web apps can be more difficult and real challenge since tester now has another parameter added to his testing matrix.

4.10 Rendering Differences

Rendering process on mobile devices, such as PDAs or smartphones, is based on many factors, and the predicting of how the app will render on smartphones or tablets is more complicated since there are so many factors involved. In some mobile phone, computational power is low by the usual limitations in processing speed. Some platforms have great support only for fixed-point arithmetic calculations and have no full support for floating-point operations or do not support them at all. The limitations of mobile memory size and the bandwidth to memory also is an issue when it is small. The limited amount of disk space or other nonvolatile memory might also impact rendering process [24].

Because of restricted battery capacity, the energy usage should also keep in mind; therefore, some mobile architecture offers power-management options. Another challenge is the small screen size of a mobile device and its various screen resolution, which has to be considered from the perspective of visual perception. In general, the differences between device manufacturers, operating systems, screen sizes and applications can all affect how the app will render on a mobile device. For different devices, there are different types of rendering. The mobile tester should ensure that mobile app looks and behave correctly on different types of rendering and screen size [24][55].

Chapter 5

Internationalization and Localization of Software

As an introduction to the topic of internationalizing and localizing a mobile application, this chapter discusses what kinds of modifications have to be made to introduce a localized mobile application. These changes influence mostly particular differences between regions, their cultures, and their locales. There are many other issues which have to be considered and converted during localization. Finally, the collaboration between people involved in the internationalization and localization processes is explained and it is shown how they can benefit from each other.

5.1 Internationalization Elements

With the growing mobile application market far beyond the English-speaking world, it is important for the application to support different scripts and data formats to reach all the potential customers. To make that happen, the most important step is to develop the application with the consideration of internationalization in mind. The below sections provide some generic information about the internationalization elements.

5.1.1 Encoding Schemes

During internationalization, various encoding schemes or code pages have to be considered. An encoding scheme is used to map the characters of our spoken, mathematical, and visual languages into computer's bit representation. Each of these schemes illustrates the one-to-one mapping between a character and a number or string of numbers. The most commonly used encoding scheme is ASCII, but ASCII is used for the relatively small character sets and does not support much more than the Latin-like written languages [58].

Unicode is a standard for representing many of the complex writing systems (e.g., Chinese) of the world. A Unicode string encoded in UTF-16 format is encapsulated by string objects.

What the user reads as a character may be represented and encoded as multiple characters in a Unicode string. Therefore, developers should use string methods that manipulate composed character sequences, not individual characters in a string. Further, they should use the proper string APIs for iteration, searching, and sorting too, and standard views and elements that render Unicode string objects correctly [58].

5.1.2 Locale and Culture Awareness

Calendar differences

The Western (Gregorian) calendar system is used in most English speaking countries, but the international software should also consider other calendaring systems in use global. For example, there are the Japanese, the Hijri, the Hebrew lunar, and the Taiwan calendars. One of the main differences between calendars is that each calendar could have a different year value. For example, the Gregorian year 2017 is the year 1438 in the Hijri calendar. The length of the year and months might also be different, as well as how they are handling leap years. Or even within the same calendar, the first day of the week might start on another day besides Sunday, based on the culture. For instance, in most of the European countries that use the Gregorian calendar, the first day of the week is Monday.

Date formatting

Displays of date are different from country to country and it is not fixed throughout the world. Although each date basically renders the day, month, and year, their displaying order and separators differ greatly. In fact, there may be many variations between regions within the same country. The names of the months and days of the week vary from locale to locale. However, there are other notable differences as well [6].

Time formatting

Like date and calendar formats, time formats are not fixed throughout the world. The time format basically displays the hour, minutes, and seconds, and their rendering order and separators differ greatly. In fact, there might be many variations between regions within the same country.

Number formatting

When displaying the numeric values, there are five main items to take in your consideration [13][58]:

1. The symbol used as the thousands separator.
   In the United States, this character is a comma (,). In Germany, it is a point (.). Thus two thousand and seventy-five is displayed as 2,075 in the United States and 2.075 in Germany. In Sweden, the thousands separator is a space.

2. The symbol used as the decimal separator.
   In the United States, this character is a point (.). In Germany, it is a comma (,). Thus two thousand seventy-five and eight-tenths is displayed as 2,075.8 in the United States and 2.075,8 in Germany.
3. The placement of the negative sign.
   The negative sign can be used at the beginning of the number, but it can also be used at the end of the number as in the Arabic language. Thus a negative six hundred and eighty-five could be presented as:
   - -685 (English)
   - 685- (Arabic)
4. Numerals shapes.
   Numbering styles vary depending on the region and might be shaped differently from one locale to another. Further, numbers in some locales might not correspond directly to their digits (0-9) used in Latin-based languages.
5. The placement of the percent sign (%).
   It can be presented in different ways: 98%, 98 pct, %98. Thus developers should not use hard-code string for the percent sign (i.e., separating them from code).

Units of measurement

Things are measured using several units and scales throughout the world. The most common one used is the metric system (meters, liters, grams, etc). The type of measurements referred to are for things such as [19][58]:

- Lengths
- Weights
- Area
- Volume
- Temperatures
- Paper sizes
- Angle notation

Therefore, developers need to make sure that if they deal with measurements, they can present them in several systems. Also, they should make sure it is clear to the use on what system is currently being displayed.

5.1.3 Input, Output, and Display

Text input, output, and display

The input, output, and displaying of data is a trivial process, however, the user enters text using a keyboard (or maybe with a voice-recognition or handwriting-recognition engine), and the application shows that text using a selected font. This evaluation of the process assumes that each user is monolingual. What if a user attempts to enter text in many different languages? In this case, the process becomes much more complex. Various languages have various keyboard layouts, and the characters of each language might fall into different character sets that need separate fonts [58]. This section covers the many ways to handle each of these areas.

Input method editors

Many Asian languages have more characters needed for writing the language than keys on a standard keyboard (e.g., Chinese). So a mechanism has to be proposed and implemented to help users in typing of all language characters by using a limited number of keys. The critical issue is that none of these components should be hard-coded in the source code of software, all of them should be stored in resource files [13]. To enable users to write in such languages, input method editor (IME) applets should be created for each language. These are presented by the platform or available through third-party vendors. In some cases, various IMEs may exist for the same language [58].

Unicode enabled

Do not expect user data will only be entered in your language characters; software should support Unicode scheme. Users should also have the ability to manage extended characters in all segments of your product, such as in document text, file names, server names, usernames, and generally, all text strings used in any way throughout your product.

Case conversion

Converting characters from lower to upper case (or vice versa) can cause many problems if the character is not present in the English ASCII code list. Some languages do not have a one-to-one mapping between their uppercase and lowercase characters and some characters have various mappings based on the language in which they are used. On the other hand, most non–Latin-based languages do not even use the lowercase and uppercase concept, as in the case of Chinese, Japanese, and Korean; Arabic, Farsi, and Hebrew; as well as Thai.

Sorting and string comparison

For each language, there is a different set of rules for how strings of language should be "sorted" or "collated" into an ordered list. Usually sorting tool in most applications based on ASCII code order. But this is not valid for all languages. User expectations for those lists may vary based on where the information is introduced, such as dictionaries, spreadsheets, and tables. Sorting and matching processes of strings are language-specific. Even within the Latin-based script, there are various composition and sorting rules. Therefore, you should not depend on code points to perform correct sorting and string comparison. Because each language has uniquely sorting rule and it tends to be complex [58].

Line and word breaking

For complex scripts, the word and line breaking introduce a distinctive issue when a multilingual script is to be parsed or presented. Latin script uses some straightforward rules for word and line breakings, such as breaking a line using a space, tab, or hyphen. On the other hand, in some languages like Thai and Khmer, words render together (with no space between letters that end a word and those that start next word). This makes word breaking process in these languages a more complex because syntax rules need line breaking on word boundaries. Therefore, the word breaking process in such languages depends on grammatical analysis and on matching the word in dictionaries during text rendering at runtime. Other languages also have rules of their own. In contrary to the most Western written languages, Chinese, Japanese, Korean, and Thai do not necessarily show the separation between words by using spaces. Although the Thai language does not implement spacing between words, it still needs lines to be broken on word boundaries [58].

Complex scripts awareness

All language versions of Pocket Code are enabled for all supported languages, thereby empowering applications that use Unicode as their encoding model to manipulate mixed text from any of the supported scripts. For instance, in Pocket Code you can render text containing English, Arabic, Greek, Hindi, Japanese, and Thai text all at once. Among these scripts, there are several that need special processing to display and edit since the characters are not laid out in a simple linear progression from left to right, as most European characters are. These writing systems are referred to as "complex scripts." To make sure that the application works properly for complex scripts, especially when displaying typed text, do not output characters one at a time! Store the text in a buffer and then render the whole string [59].

5.1.4 Isolate Localizable Resources

Resources files containing hard-coded localizable elements are the prominent problem to deal with. Translators cannot easily translate the source-code files, especially if the code is

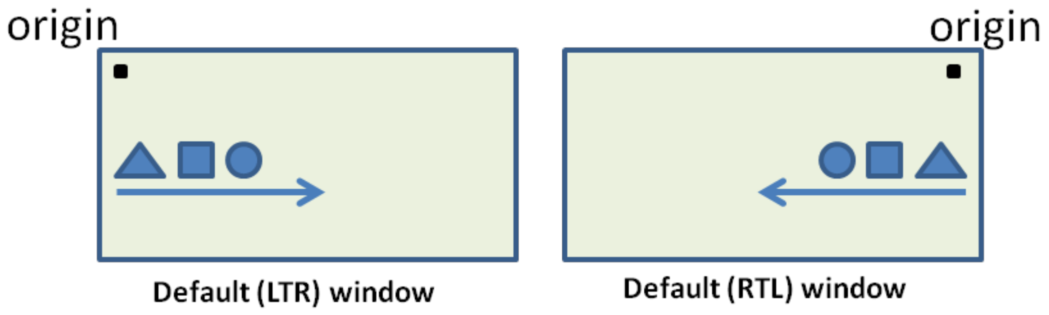


Figure 5.1: Coordinate transformation.

continually evolving. Many translators are not programmers and might remove essential details, such as closing quotation marks or semicolons, or they might translate programming keywords as well as strings [21].

A critical issue is that none of the UI element's text, colors, images, or styles should be hard-coded in the source code of the software (i.e., separating them from code). All of them should be stored in resource files; the content of resource files can be retrieved by the software at runtime to supply these elements to the users in their local language.

5.1.5 Mirroring Awareness

For bidirectional languages, not only does the text alignment and text reading order render from right to left, but also the UI elements layout should follow this natural direction of rendering from right to left. Of course, this layout change would only apply to localized bidirectional languages. Mirroring gives a consistent bidirectional look and feel to the UI. Mirroring is, in fact, a coordinate transformation and in RTL layout the Origin (0, 0) is in the upper right corner of a screen as shown in Figure 5.1.

5.2 Localization Elements

Now that all elements related to internationalization have been introduced, the following section gives an overview of what to consider when localizing software. There are many elements that are modified in both software and content localization. These elements include text, layout, graphics, color, keyboard shortcuts, fonts, as well as the locale data and character sets [6][13][58].

5.2.1 Text

Text translation is the most important aspect of localizing the software UI. The translation should communicate in language, phrasing, and vocabulary that is a natural, accurate, succinct, complete, and correct translation. By the way, the language should be clear, familiar to the end user, and suitable for the culture of the target locale. It should pursue conventions used in the target locale such as accustomed symbols, punctuation, formatting, and typography. In particular, menus, dialog boxes, buttons, text views, messages, title bars, and tool tips are some common UI elements that contain text. User manuals and help files typically need heavy translation efforts.

Translation does not always produce in a one-to-one correspondence between the original and target language. A single word in English can have multiple translations in another language. For example, the word "play" in English can have different translations in Arabic, depending on whether the word is used as a verb or as a noun. Even within the verb and noun categories themselves, the Arabic translation can vary, depending on the specific context. On the other hand, particular English words might have only a single meaning in another language. In addition to semantic differences among words from one language to another, adjectives and articles sometimes change spelling according to the gender of the nouns. Therefore, when re-using a text string in different locales, localizers should ensure that the string is translated in such a way that the text is correct for a specified context.

In general, the text (as well as the graphics and other components) of the localized product should not express locale-specific geopolitical views (e.g. using country flag instead of language name) since this could lead to inconsistency in the localized product. Nonetheless, sometimes there could be undesirable consequences resulting from a localized product that is not customized to meet a locale's geopolitical views. If the views in the product are going to insult the target market, this will affect sales in that market.

Content localization

Content localization is a necessary part of adapting a product for a certain market. Content localization consists of printed documentation, online content, and Help files. Content can be actual parts of the application software or an online software by itself. Software localization includes various methods than those used for content localization. One of the most noticeable differences between localizing content and localizing software is that the Help text often consists of long consecutive text that can be translated as one big piece. For instance, in the case of an online Help website, there are usually short topics that are attached to each other by hyperlinks. Each topic page still describes a complete entity in itself that is generally understandable to both the reader and to the localizer. Thus this text can be translated from the first to the last line, sentence after sentence, without the localizer ever having to admire where a particular string might appear in the product or what it could mean. This is especially true since this text is produced to serve as documentation and is, by nature, somewhat self-explanatory [58].

Sometimes, you need to distinguish between different countries where the same language is spoken, e.g. localization (en-US, en-CA) vs. localization (en-GB, en-AU).

Furthermore, length limitations for strings are potential defects that can cause big problems. Sometimes, translating a string from one language to another will change the length of the string. For instance, a product build was broken due to the German translation for

the following string:
"Press Ctrl+Alt+Del to restart."
When translated into German, this string almost doubled in size to:
"Drücken Sie Strg+Alt+Entf, um den Computer neu zu starten."
The increased string length causes the buffer to overflow, which in turn will crash the system. The only thing that can be done for this type of problem is to make sure that you communicate to localizers how much space they have for their translation and the developer can allocate more space than necessary for English. Localized strings tend to uncover buffer overrun problems. These kinds of problems can be overcome if the buffer size is dynamic or is allowed to have the maximum buffer size.

5.2.2 User Interface Layout

Another element that needs localization is the UI layout. The localization process changes the UI layout because the length of the translated string usually increases from its original length. This string-expanding affects the size of the UI elements that render the string and potentially necessitates relocating and resizing the UI elements. The best way to ensure that the UI has a consistent look and feel throughout several localized versions is for the development team to design flexible UI with resizing and other required locale variances. In turn, the localization team should minimize any alterations to the UI, unless these alterations are absolutely necessary.

In addition, font size can vary among different languages. For example, East Asian languages usually render text in a larger font size than many other languages. As a result of the change in font size, the system resizes the UI vertically. UI controls within the East Asian localized version of the product usually result in expanded vertical spacing than those within products from other languages.

Language and screen layout of a user interface are associated with each other. Languages are read in different ways. English and other Western European languages are read from left to right. Others, for example, bidirectional languages like Arabic and Hebrew, which are read from right to left, require mirroring of the screen layout to accommodate their RTL reading order. This requires mirroring the layout of menus, text, dialog boxes, and spinner. This also affects how the eye is moved over screens. When Latin characters are implemented on a display system, attention is set on the top left corner and then the reader moves her eyes to the bottom right corner. Whereas for the speakers of the bidirectional language, they first glance on a screen layout at the top right corner and then they move their eyes to the bottom left corner. Hence, the layout of a user interface should be mirrored for all right-to-left languages localization.

5.2.3 Graphics

Users communicate with their computer's software not only with text. There are also many logos and icons which have their own particular meanings. All of the application's graphics should be culture-neutral. In other words, all controls must not include any attributes that use the idioms of a specific culture. Graphical UI components of the product need to be revised for the international customer. Such an attribute might be misunderstood by local users. During localization phase, the potential insulting or misunderstood can be easily adjusted. Also using puns in user interfaces is considered as a potential ground for trouble and should be avoided. In some countries, they might be understood correctly but surely their meanings are not understood all over the world. For instance, icons represent fingers such as an OK sign or V-sign may mean different things to different cultures. Also, the use of animals in logos can cause problems in localization. For example, pigs are considered unclean in the Muslim and Jewish cultures. These practices will help minimize the localization of graphics, which is very expensive and often causes delaying in the shipping of the product.

Icons that have a specific directional orientation such as Back, Next, Undo, and Repeat should be given a special attention for bidirectional languages in order to be locale-aware. The localization team can present feedback about the graphics elements used in the product to ensure that they are convenient for all locales.

5.2.4 Colors

Colors also have different cultural meanings that need to be analyzed in localization for mobile apps. Choosing the inappropriate color for the logo or background will not always have disastrous consequences, but avoiding them is always desirable. For instance, in Japan white is commonly correlated with mourning. In China red means happiness. In Africa, specific colors represent different tribes.

5.2.5 Keyboard Shortcuts

Keyboard shortcuts might require being customized according to the translated string that contains these shortcuts. For instance, the "F" key is selected as the shortcut for "File" in English. This shortcut should be changed during German localization to the "D" key since the translated word in German -"Datei"- does not have the "F" character in its string. When there are standard shortcuts for functions that are common in localized software products, localizers should use these shortcuts and not assign new ones.

5.2.6 Fonts

Localized products might contain several UI fonts than the original products. The font name, size, and style in menus, dialog boxes, and documents can be adjusted to customize the final UI based on user preference. Localizers will choose the best font options according to the font capabilities of the certain target language.

5.2.7 Locale Data and Character Sets

The locale is a set of parameters that express the national variations in formats of e.g. numbers, calendars, date and time, currency, units of measures, telephone numbers and postal addresses. As well as locales can significantly differ between countries even though they have the same official language. Localizing the product to suit an international audience requires using the appropriate format for the target locale. Here are some examples of differences in locales.

Displays of money amounts differ from country to country. In the U.S. the dot '.' is used to separate the currency's whole and fractional units (1,234.56). In Germany instead of dot the comma ',' is used (1.234,56).

The display of date and time can be different, there are two systems to display the date and time, some countries use the 12-hour system (with "AM" and "PM") and others use the 24-hour system, where e.g."16:00" means "04:00 pm".

In addition, the format of date displays differs, e.g. the order in the U.S. is month/day/year and in Arabic day/month/year and in some countries the character for separating the numbers can be different e.g. day.month.year.

Units of measurement have to be adjusted, as well. The U.S. uses the English system of measurement, while most other countries in the world use the metric system.

Some instances of locale data values can be saved as part of the resources in order to initialize variables such as the default character set or the default date and time formats. If the developers want to store these values, localizers should customize the locale value strings according to the format of the target locale.

5.3 Formal Definition

This section introduces the formal definition of internationalization and localization processes of a software application. In the first place, all steps of the internationalization phase are defined. Next, all steps of the localization phase are presented. At the same time the cooperation between developers, users and translators are covered. The diagram in Figure 5.2 shows an initial overview of the internationalization and localization processes and how elements are related to each other.

5.3.1 Definition of the Internationalization Process

The internationalization process starts right in the beginning when software analysts are preparing software requirements for the application to be developed. So this process is almost the same as in other software requirements engineering processes. Actually, if the development team does indeed want to build localizable software, they should make a clear decision about how much they want their software to be localized in the following stage. The more they need to localize the more they should to internationalize [60].

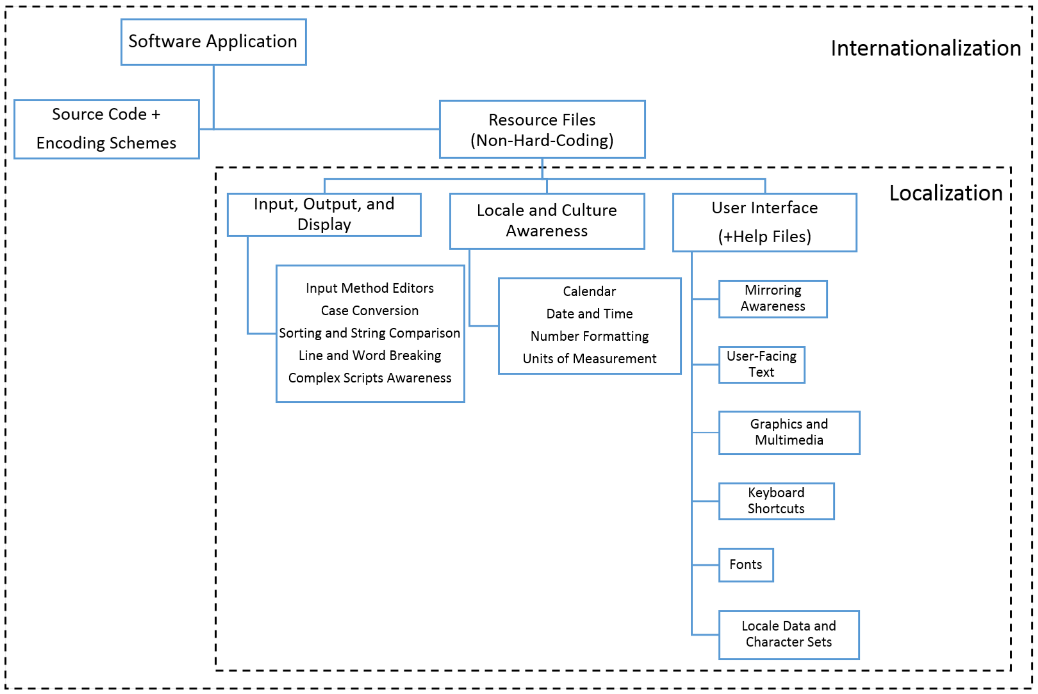


Figure 5.2: Process and elements diagram of I18n and L10n [58].

Without a doubt, it is preferable to internationalize the application's features as much as possible or as the company budget and time allows. Experience in software development has demonstrated that software development projects have been extended in various ways during their maintenance phase as opposed to initial plans. To adapt another market many sudden modifications in requirements or customizations in functionality cause those confusions in software development. In the first place, it was planned to sell the product only in a specific number of markets, when unexpectedly a vast order occurs for this software from other markets. Then it would be much more troublesome to adapt this software to those markets [60].

The critical step in developing internationalized software is to consider implementing support for internationalization issues and to make sure that all elements described in section 5.1 are non-hard-coded and stored in resource files. At the same time modules for data processing functionality should be integrated. So certain modules would be exchanged easily with others having slightly different functionality [6][60].

The functionality for sorting and collating would be an example because sort sequences vary in some languages. In the English language, for example, the sorting sequence based on the ASCII representation of a letter and the uppers case letter always sort before lowercase letters because of the ASCII character ordering. The standard order of the basic modern ISO basic Latin alphabet is: (A-B-C-D-E-F-G...). While in the Arabic language, the standard order is (... خ - ح - ج - ث - ت - ب - أ), so the sorting functionally for the Arabic needs to support a special code page by exchanging the certain modules of sorting and collating with Arabic one.

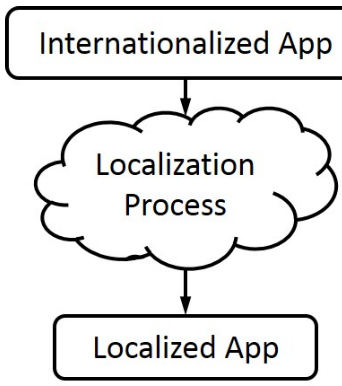


Figure 5.3: The localization process.

Furthermore, locales can be implemented as modules. Depending on the programming language that the developer wants to use, locales do not need additional development. Java, for example, has a huge library that supports locales where all necessary features are already implemented. At this stage, the development team should consider also what kind of encoding scheme is to be used. Unicode introduces the most freedom to developers but requires more memory resources due to 2-byte and 3-byte character sets. The user-facing texts which are used to communicate with the users should be stored in values resource file and separated completely from the source code [60].

5.3.2 Definition of the Localization Process

The first step to reach the global market is to internationalize the applications; the next step is to specify the localization process. Often applications are distributed to many countries and regions. Every country or region has its own language and culture; accordingly, they should be localized to meet all demands of local users. So what is behind the cloud in Figure 5.3 is the localization process that guarantees their usability and robustness. There are three major steps to localize application successfully [6]:

1. Translating all user-facing text strings in the application to specific language before it is displayed to the user.
2. Customizing graphics, icons, colors, fonts, styles, menus and if necessary the text rendering of each user interface element for different cultures to display text correctly and matches the intended specifications and all expectations of local end users.
3. Adding a locale that guarantees that all country-specific formats are shown correctly.

Because this process is much more involved, localization requires the integrated of expert translators who are familiar with the app domain and target locale. It is not recommended to simply use machine translations to localize the apps. Only native translators who are familiar with the app area, speak the native language and has widespread expertise in the Arabic region will make the app appealing to customers and subsequently cause more downloads, revenue and positive rating for the app. Actually, the literal translation will almost always minimize the value of app with regards to the user experience and it is not as succinct, complete, and correct as when a native speaker translates it. Your branding and marketing message will inevitably get lost in translation. For mobile apps, this is where the necessity of localization comes in. The question is if an app publisher wants to get traction in several countries in terms of a total number of downloads and revenues, how important is it to localize one's apps?

The content should be translated appropriately. However, the content should be translated into the language of the target locale. Localizer should make sure to perform translation process with language professionals who can best recognize regional dialects or slang idioms that will resonate with the target audience. These specialists will basically keep things from getting lost in translation, or cultural errors from incidence. Sometimes, translating string from one language to another will change the length of the string, so more space should be allocated than necessary for English, to accommodate most other languages up to 50 percent more is normal when translated into another language.

Localizing an app is more than just performing a basic translation. It needs to use the right formats for date and time, measurement units, the right currency, use the payment gateways the target locale is most used to, use the appropriate layout and text direction, and much more customizations [59]. If not performed properly, localizing the app makes it vulnerable to misleading translations and graphics that might be culturally inappropriate or direction-sensitive. This is where the importance of textual contents comes in. Translating the app should be done with the cultural particularities of the target locales that only who are very familiar with the application domain would know. This is why human localization that is performed by a local of the region is critical.

Customizing graphics, icons, colors, fonts, styles, and menus for other cultures and languages should be accomplished by a person who is familiar with the cultural and linguistic status of the particular locale. It is even better to let this process be accomplished by a variety of specialists, including engineers, and graphics designers who know best about software environment and cultural differences. Graphics, which are direction-sensitive, and maybe some styles or colors of the user interface have to be modified [59][60].

The direction of text becomes a critical issue when the application has bidirectional languages texts, which are written and read from right-to-left. Hence, all user interface elements should support bidirectional languages; the text alignment and text reading order go from RTL. The feature for switching text rendering directionality for bidirectional languages should be done pretty easily [6][13].

The system formats are used to specify dates, times, numbers, currencies, and other values that can be changed by locale; i.e., format all data according to the locale. If there is a specific format that based on assumptions about the locale of users, the problem will arise when the user changes to the other locale. At the same time adding a locale is only required if the programming language the software was implemented with or the operating system the software is running on do not support a "global" locale setting by themselves.

In summary, internationalization and localization of an application need more than just translating UI string. It is a much more complex process, the essential steps for which should be considered from the very beginning of the application design and development.

Internationalization and localization issues should be considered constantly when creating the base of an application. This will save a great amount of time and money when localizing it into different languages.

5.4 Why Localizing Mobile App Is a Must and so Important?

Mobile applications are getting developed in many areas, including business, gaming, health, photo editing, and engaging in social networking. According to the statistics website Statista, a number of free mobile app downloads worldwide (Apple's App Store and Google's Android Market) is forecasted to reach 253.91 billion downloads in 2017 [39] and paid app downloads are predicted to be 14.78 billion downloads [40].

Developers of the mobile app are always looking for ways to raise their app downloads and rates, if mobile app company wants to increase its revenue, recent data shows that company should have to think bigger than just the English speaking market. Every software company wants to reach as many potential users of its mobile app as possible, the app needs to talk to them in their native language. Enable them to get the best experience in the language of their choice. Even the customers who speak English mostly prefer using a smartphone in their native language and are more willing to buy if they can perform that in their mother language, therefore, localizing of a mobile app is a must.

When companies are planning to release a new mobile app, localization can be a low priority on their to-do schedule. Why? Because of most of the time, there is almost no localization consideration until the later phase of the development cycle. That is why localization aspects will typically be a lower priority and when the product gets hectic, those issues will likely be neglected altogether.

Even though the necessity of localization requirements and translation have been proven time and time again, there are still a great number of traditional mobile app publishers and developers do not take localization seriously in their consideration or do not prioritize localization tasks properly. For example, creating apps only in English prevents app star power from reaching global markets. It is completely fine to start developing a mobile app in one language, but if the company wants to reach the huge international market, it cannot stop there.

Figure 5.4 shows abandonment rate for the mobile app because of insufficient localization in selected countries as of March 2014. During the survey period, it was found that 48 percent of app users in the United States had stopped using an app due to lacking localization [61].

Localization is now the main factor in moving an app into new markets. Whether you develop mobile apps for web surfing, gaming, or engaging in social networking, it is important to consider all the localization aspects of which enhance the dominance of your app to localized markets. Localization of mobile apps has a direct effect on download volume and revenue. Trying to compete in global markets but not sure how to reach target both quickly and effectively? Pretty sure the answer is app localization.

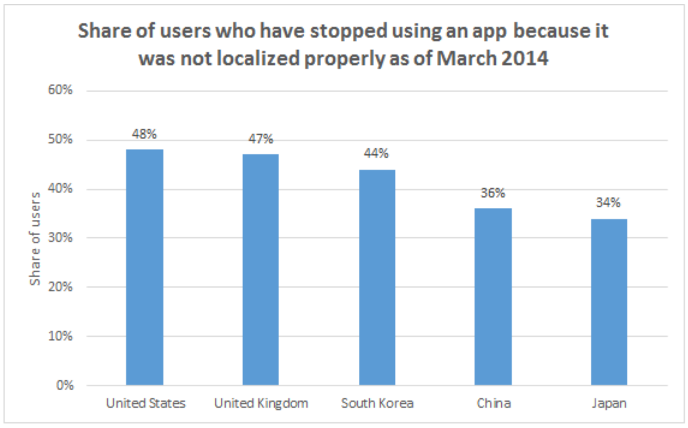


Figure 5.4: Abandonment rate for the mobile app because of insufficient localization. Sources: Admob; Google [61].

There is a clue indicating that the key to reaching more users is via localized apps, versus the un-localized ones. If the company is looking to get more app downloads and drive adaptation in both new and emerging regions, localization is an expansion process it cannot ignore. Perhaps it is one of the most critical issues of first reach to new markets and localize app's UI and source code in the top languages that are emerging markets for downloads.

There is no question that the internationalization and localization processes value of apps is going up and up. Yet, in a worldwide where language plays the critical issue to taking a product and tailoring it to a specific locale, translation is a must. Moreover, the value of an app is not what this app have to introduce, rather what audience can get from it. Hence, it is not enough just to have an app in the global app store; customers need to be able to get the full value of an app in their mother language.

5.5 The Importance of Localization Process in App Stores

It is clear, the higher an app ranks in an app store, the more likely it is visible to potential customers. Competition is high as more than 6 million apps have been distributed in leading app stores by March 2017 [38]. As of that month, Android users were able to browse and select between 2.8 million apps. Apple's App Store remained the second-largest app store with 2.2 million available apps. Getting your app downloaded is probably the biggest challenge faced by mobile app developers. Thus, marketing your app is critical to be discovered among this flood of applications. App Store Optimization (ASO) defines the process of optimizing mobile application in order to get a higher rank in app store's result report and higher downloads. General guides for ASO mainly focus on the aspects title and description, keywords, total number of downloads and ratings.

5.5.1 App Store Optimization

After a mobile application is localized, app publishers are challenged by ASO. If the business companies are indeed interested in getting more traffic to their mobile apps there is no neglecting or even getting around ASO. Being exist in app stores is one of the most challenging aspects of mobile apps. However, this challenge can be actively overcome concentrating on app store optimization. Localization is an important part of both ASO and app marketing principles as it gets the attention of customers in new markets. Localization promotes the app visibility and brand recognition and the company will earn the rewards on the long run. App store optimization for multilingual apps is not a negligible task, this task should cover translating the app title and description, keywords and metadata, screenshots, previews, and videos [62].

Localized title and description

Translating an app name can be helpful for potential customers in order to better understand the app's objectives at first view since app descriptions are not shown on search result pages. However, app publishers might suggest that the app name should not be translated due to branding issues. Developers can add attractive text and embed keywords in app title to better help target users understand what an app is all about. To translate the app description, translators can use a simple text that might be acceptable for some of the users. However, the developer can enhance and use translation to generate a significant value to new potential customers.

App publisher should think bigger than just the U.S. market and how much additional users they could get by localizing the app description. Even though some of the users might speak English, releasing the app in many languages will raise app downloads and rates. Translating the app description is a must if the target app is available in several languages, and is a probably a good idea to do just so that potential users will better understand the objectives of the app before downloading it.

As soon as developers localize their own app description, they should start closely watching downloads rates in the correlated languages and countries. If they detect a big increase in downloads, it is time for them to start planning to localize the app itself. Introducing app's descriptions in Arabic will likely drive more downloads, but localization will drive the next level of adaptation.

During the translation process for the app description, publishers should try to avoid the following issues:

- Keyword stuffing: simply fruitless and will scare users off
- Typos and syntax errors
- Using technical terms
- Being untruthful (the result will be in the form of bad ratings)

Certainly, the translator should adapt app's description to target local customer base. It is recommended to include localized keywords in the app description. The app's translators should be provided with a collection of translated and optimized keywords. Then, they can suggest an engaging app description in their mother language including local terms, slang and cultural expression to match certain locales.

One thing publisher has to consider it is that for users who are visiting the app web page on a desktop computer or iPhone/iPad, just the first three lines of the app store description are directly visible, with the app icon and the first one or two app screenshots. In all situations, on the real devices, the app description is beneath the screenshots and users have to tap the "More" button to navigate the remaining of it. Therefore, the publisher has to try to put an essential information there, like the best app pitch, for example, a sentence that explains clearly what the app does and why it is better than other ones. Actually, a huge number of users do not surf the full app description before making their choice. Accordingly, the most important stuff should be on the top.

By using app's title, users can do a search for any app in any language, but keywords for searching purposes are translated and show results that are relevant to each particular locale. By translating the app name and description and adding region-specific keywords (that have been optimized for each locale), businesses can make sure that their target audience will find the app easily. Simply translating the app market description can be a quick way to find the app. For instance, it is possible the potential users in Japan do not know an app available on the app store since they simply do not do searches in English. If sufficient customers download it, the app rankings are one of the top downloaded apps in its class, therefore enabling more discovery opportunities.

Localized keywords and metadata

The most necessary part in keyword's localization is succinct and correct translation according to strategic purposes. When it comes to selecting the right keywords for the apps, large volume and low competition are the best. The translators should choose their keywords carefully in regard to app store optimization in order to make the app at the top of search results. Be aware of the volume of keywords depends on the volume of supported locales and languages.

Conceptually, a total number of downloads and revenue volume give developers a more precise vision of the markets that it is recommended to localize their apps in. The publisher should make sure that the app's metadata is localized to achieve the two goals: downloads and revenue, in each of the top markets and locales. App publisher should keep in his mind to optimize the app title, market description as well as localizes log and search keywords based on native language and cultural peculiarities.

Localized app screenshots and videos

A picture can express thousands of words. Nothing keeps the app as visible as screenshots and videos. Naturally, the human's eyes always prefer to focus on visuals. Besides the app description and application ratings, application screenshots are what has to convince customers to download or buy an app. In fact, the most important issue for mobile software distribution platforms (App Store, Google Play, etc.) that they have to consider it is: if the app screenshots are very bad or disagreeable or if publishers do not include the proper ones, users will not download the app. Like always, it does not mean that catchy screenshots on the app store page will get the app huge downloads but it is necessary if mobile app company want to raise their app downloads and rates, and increase its revenue. Just like for app icon.

The publisher cannot get good screenshots without a good design. Obviously, a good design for the app is something publisher should consider it from the initial. It covers a great user experience, simplicity, and awareness for details. Some publishers do not upload all the app screenshots and that is, of course, a fault. It is recommended to include as many as they can. Probably the app developers designed a great splash screen but it does not explain much about the target app. The first screenshots that should be seen on the app store by potential users have to describe what the app does and why it is so great. If mobile app company wants to release of such apps on a worldwide scale requires them to be made available in many languages. Consequently, it is essential to localize app screenshots and videos as well and upload screenshots for each of them.

Undoubtedly, to guarantee that the localized app matches all expectations of local users, and guarantee that the product is culturally congruent to local conventions, it is essential to localize app screenshots and videos as well. To provide potential users with attractive graphics in their mother language, the app publishers should consider the following issues:

- If the app screenshot contains any text it should be placed on external resources and retrieved later to be translated by specialists. Once the app is fully translated publishers can easily take screenshots of the localized version.
- When the graphical picture expresses any country specific location or icon, publishers should consider to change it. For instance, when the publisher wants to localize the app for the Arabian market, he should think about replacing the picture with something typical to Arabic people and culture.
- Mobile app publisher should bear in mind to type some explanations on app screenshots to make sure users that do not know anything about the app can get it at once. It could be a great value to convince potential users to buy or download the target app.
- Sometimes, app revenue is the major purpose for releasing a mobile app, but the app marketing for enhancement in sales requires the localization activity. Further, it is important to consider what languages will be necessary to localize the app first, then go down the list and fully consider all localization issues to guarantee that the localized product meets a local user's expectations in terms of language, features, and user experience.

There are numerous ways to decide which languages app should localize into or which languages are the most relevant for the localized app. Mobile publishers need to gather some statistics for different countries such as GNP, population, mobile phone penetration and mobile platform market share. If the app is already localized into English (or any other language), publishers can survey the app store statistics to get information about where the app is receiving more downloads and find which locales are interested in the app. Finally, all the mentioned issues indicate that localization process will go a long way to expanding the app total number of downloads, ratings, and revenue internationally.

5.5.2 Reaching New Markets With Arabic Localization

In recent years, mobile apps have become an integral part of our daily activities, apps have become pretty much effective tools for both companies and customers, once an app is localized, businesses can dominate a much wider market than ever before and reach the global market. If businesses are planning to enhance their revenue volume then localizing the apps for the growing demand in the Middle East and North Africa (MENA) market is one of the regions they can consider to raise a number of downloads and enhance revenues, recent statistics have proved that localizing the mobile app to the Arabic world could be crucial if companies want to support their business.

Why is it substantial to localize the app to the Arabic world? A few reasons:

- While smartphone usage has raised in all countries, the Arabic world is one the notable growing regions for smartphones. According to a recent survey from Statista, there is a flow of growth in the usage of smartphones in the MENA region [63].
- There will be a rapid increase in smartphone usage in the next years throughout Saudi Arabia, Egypt, Jordan, Lebanon, Kuwait and the United Arab Emirates; Further, the number of smartphone users in Saudi Arabia is forecasted to reach 25.3 (millions) in 2021 [64].
- The Middle East and Africa (MEA) is one of the world's most lifelike and fast - growing mobile markets - and one of the biggest. In 2015, eMarketer projected that more than 789 million people in the Middle East and Africa will have at least one mobile phone in 2019. Smartphone usage in the Middle East and Africa is rising more quickly than eMarketer projected in the past years. While in 2015 fewer than 18 percent of mobile phone users in the region had such a device. In 2019, that share is anticipated to reach 22 percent, as the number of residents with a smartphone approaches 174 million [65]. If the forecast is confirmed to be accurate, then that is a potential market of 789 million customers that the business can market its app to; such a large market cannot be neglected, which makes Arabic localization essential for increasing downloads and revenues.
- The Arabic region is native land to many prosperous companies and extremely rich investors. For this reason, many entrepreneurs and businesses are enthusiastic to found links in this market in order to enhance their fortunes and maximize the opportunities of getting investment and one way of achieving this is via mobile apps.

Chapter 6

Challenges of the Arabic Localization in Mobile World

The Arabic language is ranked as the 5th language in the world in terms of a number of native speakers, and it is also one of the six official languages of the United Nations. It is considered as the first language for more than 250 million people and about 280 million people use Arabic as their second language [10]. Arabic is the main source of vocabulary for languages such as Bengali, Kurdish, Pashto, Persian, Swahili, Urdu, Turkish, Malay, and Indonesian, as well as other languages in countries where these languages are spoken. In addition, Spanish and Portuguese have great numbers of Arabic loaned words [13].

6.1 Internet and Mobile Penetration in the Arab World

Arab countries are highly affected by the Information Communication Technologies (ICT) which impact their economies and lifestyles. According to the research conducted by the Internet World Stats, in 2016 more than 168 million people used the Internet in the 22 Arab countries, representing nearly 5 percent of global Internet users [66]. Despite the growing of Arab Internet user, the number of localized Arabic web sites and websites with original Arabic content remains low. Jordan occupied the first rank among Arab countries as the major contributor to the online Arabic content on the Internet which amounts to less than 3 percent. However, about 75 percent of the Arabic content on the web pages comes from Jordan, according to a recent survey conducted by the International Telecommunication Union (ITU) [67].

Arabic text is rendered from right-to-left while numbers are read and written from left-to-right, this feature has limited the availability of online bidirectional text platforms. Due to the shortage of bidirectional text supports, most Arabic users are forced to use English or Arabizi platforms, a form of Arabic conveyed through Latin characters and European numerals. This fragmentation of language dramatically complicates the process of adapting, integrating, and indexing online Arabic content for search engines.

Besides to opening new markets, this critical number of Internet and mobile users in the

Arab region play a major role in establishing of ambitious service delivery initiatives; such as electronic or smart government, mobile government, smart cities and technology which enables citizen engagement initiatives. The Internet growth will highly affect the economic development in the Arab region.

6.2 Language Is a Critical Impediment for Arabic Internet Users

According to a Google spokesperson, in 2016, Arabic has ranked the "eighth language in terms of growth and usage on the internet". Therefore, there is a huge gap between the number of Arabic speakers and the volume of content available online. The percentage of Arabic speakers or people who would benefit from Arabic content is 6-7 percent of the world population [68][69]. As an example, 0.9 percent of Wikipedia articles are in the Arabic language.

6.3 Forms of Arabic Language

Written language in Arabic is more complex than the other semantic languages. There are three forms of Arabic: Classical or Qur'anic Arabic, Modern Standard Arabic and Spoken or Colloquial Arabic [66].

6.3.1 Qur'anic or Classical Arabic

Qur'anic is based on the Medieval dialects of Arab tribes. It belongs to the supreme group of languages. It is one of the last surviving supreme languages along with Hebrew, Amharic, and a dialect of Aramaic. Classical Arabic is the language of the Holy Qur'an (Muslim's Holy Book). Primarily, it is learned for reading and reciting Islamic religious texts and etiquettes. It is the language of Early Islamic Literature [17] [66].

6.3.2 Modern Standard Arabic

It is a simplification form of classical Arabic. It is the teaching language in schools and universities. In addition, it is used in news media, formal speech, literature, science and technology and for governmental and administrative purposes across North Africa and the Middle East. Therefore, there is only one form of writing of Arabic and it is shared and common universally [66].

Modern Standard Arabic is one of the six official languages of the United Nations. It is the language of oral and writing communication between literate and educated Arabs from different countries. The term "Modern Standard Arabic" is mainly used in the Western World. Many Arabs do not distinguish between classical and modern standard Arabic.

While Modern Standard Arabic is the only form of writing in Arabic, there are several forms of spoken Arabic.

6.3.3 Colloquial or Dialectal Arabic

They are different dialects of Arabic spoken in the regions and countries of the Arab World. Some of these dialects are similar while others are not. People speaking different dialects can sometimes understand each other or they can use Modern Standard Arabic to interact. However, the mother tongue language of most Arab-speaking people is colloquial Arabic, while Modern Standard Arabic is taught at schools and universities [66].

Particularly, Arab speakers use a dialect in their everyday life interaction, but when they face a situation calling for greater formality in language, Classical Arabic would be used in religious situations and Modern Standard Arabic in other situations. There are four main dialects: the Eastern dialect, spoken in Syria, Jordan, Palestine, Lebanon, the Arabic Gulf dialect spoken in Saudi Arabia, Kuwait, United Arab Emirate, Qatar, etc., the Egyptian dialect, and the North African dialect, spoken in Morocco, Algeria, Tunisia, etc [17].

Developers and translators are often faced with the critical question: "Which Arabic form should we translate into?" There is only one form of written Arabic. Translation is carried out into Modern Standard Arabic, which can be identified and understood by all educated Arabs. There are slight differences among regions' locales such as calendar types, numbering formats, weekends, naming conventions, etc. With the Arabic language, which is spoken in more than 22 countries, a minimum number of locales are from four to five Arabic locales which are needed to accommodate various user settings within the Arab world. The following is recommended as a minimum:

- The United Arab Emirates
- Egypt
- Lebanon or Jordan
- Morocco or Tunisia

Note, if there is a possibility to choose only one locale for the whole Arab world then the better is to choose the UAE.

6.4 Characteristics of the Arabic Writing System

This section covers the major characteristics of the Arabic and other bidirectional languages. In the area of app localization, the Arabic language still remains one of the most challenging languages [17]. Although the Arabic language is right-to-left, it is also bidirectional, which means that numbers and text in Latin based characters will display left-to-right. The Arabic language is cursive, bidirectional, and context dependent [6][17].

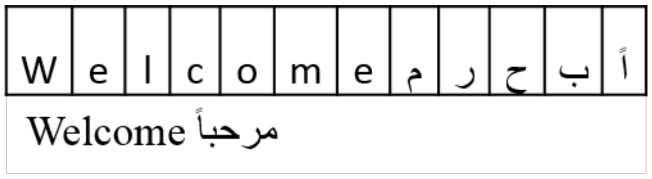


Figure 6.1: Arabic text (bidirectional) where the logical order in the first row and the visual order in the second row are not of the same sequence of characters.

Bidirectionality and character reordering

Arabic scripts (used for languages such as Arabic, Persian, Pashto, Urdu, and other) and Hebrew scripts (used for Hebrew, Yiddish, and others) are read from right-to-left, further, numbers are read from left to right with the highest order digit on the left side. This leads to a mixed direction of text parts. However, it is commonly expected by readers of such languages for the scripts to be right-aligned [6][17].

For these particular scripts, the logical order (the order in which the user enters text with a sequence of virtual-key inputs) and the visual order (the order in which characters are rendered on the screen) are different in most cases (see Figure 6.1). Character positioning and a hat character movement in bidirectional context, in which RTL characters and LTR characters coexist, are the significant challenges to solving when dealing with RTL scripts. A bidirectional context can be a mixture of Latin and Arabic or Hebrew text, or it can include Arabic and Hebrew characters with numbers that have an LTR characteristic in Arabic and Hebrew [6][59].

6.4.1 Contextual Shaping

The Arabic character font is cursive which means that the letters are connected together like English handwriting. For the Arabic and Indic families of languages, a character's glyph (that is, all the character's different possible representations) can form in four different shapes depending on the glyph's position in a word and the surrounding characters. Whether the letters are in initial, medial, end or isolated form, they will take on different shapes [70][71]. An example of the four different shapes of the Arabic character "ض" (Daad/ d) is illustrated in Table 6.1.

Initial Form	*Middle Form*	*End Form*	*Isolated Form*	*Unicode*	*Name and Pronunciation*
ضـ	ـضـ	ـض	ض	0636	Daad/d

Table 6.1: Four different shapes of the Arabic character "Daad/ d.

The six remaining letters can only be on two possible shapes as they can be only connected to but not from. An example of the two different shapes of the Arabic character "و" (Waw/ w, ou, uu) is illustrated in Table 6.2. The shape of the Arabic alphabet is depending

Initial/ Isolated Form	*Middle/ End Form*	*Unicode*
و	ـو	0648

Table 6.2: Two different shapes of the Arabic character "Waw".

on context. Furthermore, Upper and lower cases do not exist in Arabic, and seldom abbreviations and acronyms are used. The challenging part of contextual shaping is that, for all the different glyphs, there is only one defined code point in different encoding standards. Layout and rendering mechanisms should determine (at run time) the proper glyph to be used from the font tables, depending on the context.

6.4.2 Ligatures

For Latin script, there is a direct one-to-one mapping between a character and its glyph. (For example, the character "s" is always rendered by the same glyph "s"). For complex scripts, groups of two or sometimes three characters linked together to form a new glyph independent of the original characters [70][71]. An example of the "lam-alif" ligature is illustrated in Table 6.3.

Characters in Arabic	*Contextual Unicode values*	*Isolated From*
ل +ا	0644+0627	لا

Table 6.3: "lam-alif" ligature.

6.4.3 Diacritics

In Arabic, vowels are represented by marks positioned above or under the basic letters. These marks are called diacritics and in Arabic, they called "Tashkil". There are eight major diacritics in Arabic. There are single or double diacritics. The presence of diacritics changes the pronunciation and sometimes may change the meaning or tense. Also, they help the language speakers to distinguish between words of similar spelling, the below table 6.4 shows the 8 major diacritics in Arabic language [66][70].

The use of diacritical signs is optional in Arabic written standard, while they are generally present in the Qur'an or in religious literature. They are also used in teaching children to read. The positioning of diacritics can be affected by ligatures.

6.4.4 Numerals Shapes and Dates

Numbering styles vary depending on the region. Arabic numbers may be implemented by either Hindi digits or Arabic digits as shown in Table 6.5. In North Africans, the people use the Arabic formats, which are the same as the ten characters used in the European numbering systems, while in the Middle East, the people use the Hindi formats. In both

Diacritics	*Unicode*	*Example*
Fathah	064E	اَ
Dammah	064F	اُ
Kasrah	0650	اِ
Tanween Fathah	064B	اً
Tanween Dammah	064C	اٌ
Tanween Kasrah	064D	اٍ
Shaddah	0651	اّ
Sukun	0652	اْ

Table 6.4: The eight major diacritics in the Arabic language.

cases, numbers are written from left to right. Both the Western (Gregorian) and the Islamic lunar (Hijri) calendar systems are used in Arabic region. Western and Islamic months' names can both be written in Latin or in Arabic styles [66][70].

Arabic digits	0	1	2	3	4	5	6	7	8	9
Unicode Value	0660	0661	0662	0663	0664	0665	0666	0667	0668	0669
Hindi digits	٠	١	٢	٣	٤	٥	٦	٧	٨	٩

Table 6.5: The Arabic and Hindi digits.

6.4.5 Justification

Justification of Latin script is done by inserting inter-word glue, which can be stretched or shrunk to some extent and adding additional spaces to the text and inserting these spaces between words. Because of the cursive nature of Arabic writing, the text justification logic in Arabic is quite different. Justification has traditional processes in Arabic typography since it is inspired by calligraphy manuals [66]. In Arabic, justification is done by stretching the last letter of specific words in a line. This stretching process is called a Kashida or Keshide and looks like a horizontal straight connecting line.

6.4.6 Punctuation

The Arabic language mainly uses the same punctuation as the Latin-based scripts such as period, exclamation mark, and colon, although it is less standardized than in other languages, as English or German. While the exclamation mark, period, and colon are similar in both Arabic and English, the question mark, comma, semi-colon and percent sign are presented in a different style that adapted with right-to-left mode as shown in Table 6.6. Moreover, all brackets and parentheses should look the same as in Latin script [71].

In Arabic	*How it is used*
Question mark (؟)	It is just reversed and faces rightwards
Comma (،)	It is "upside down"
Semi colon (؛)	It is "upside down"
Percent sign (٪)	The zeros on each side of the slash are only replaced by dots since zero in Arabic is represented by dot

Table 6.6: The Arabic punctuation standards.

6.5 Challenges of Linguistic

During the last few decades in the whole world, a huge of new scientific, technical and business terms are created. It is evaluated that around 17,500 new terms are formed every year in different fields. Many of these terms have no Arabic equivalents or parallel terms. This shortage of standard terms for common Western terms in different fields is a challenge for translators [66].

Frequently, the Arabic translators face a difficult task of "Arabizing" different scientific, technical and business terms during the localization process. Therefore the translator needs to choose terms or expression from the Arabic scientific heritage. Moreover, the translators have an option to choose among other processes like transcription, literal translation, calque, cultural equivalent and translation label.

In the Arab world, there are many agencies have the responsibility to produce equivalent technical term for the original term, some of these agencies are official, some of them are not. These agencies can be categorized into three groups. The first group includes the National Arab Language Academies of Cairo, Damascus, and Jordan, and the Iraqi Scientific Academy. In the second group, several research institutes are also contributing in the production of terms such as Institute for Studies and Research for Arabization in Morocco, the Kuwait Research Institute, and the Arab Development Institute. Finally, the third group includes Pan Arab organizations and professional associations which usually produce terminology in their specialization field (standardization, administration sciences, agriculture, oil, physics, chemistry, mathematics, etc) [66].

Outside the Arab World, the United Nations is considered as the largest producer of Arabic terminology. The United Nations has generated 'ArabTerm', an online database for a multilingual terminology, which stores the translation of 42,000 terms, generally used by the UN, in Arabic, English, French, and Spanish. UNTERM is the United Nations terminology database which stores 85,000 terms related to a hundred fields and is available in Arabic, English, French, Spanish, Russian, and Chinese [66].

6.6 Cultural Considerations

Most Arabs are Muslims. Special consideration must be given to cultural and religious sensitivity during the localization process (especially in graphics and images), as a result of not all icons, symbols or images have a universal meaning. Translators may also have limitations regarding the content of the files they are willing to translate. The work days in most Arab countries from Sunday-Thursday, therefore, the Arabic people do not work on Fridays and Saturdays. Therefore, their work week starts on Sundays.

6.7 Differences Between Arabic, Farsi and Hebrew

Some examples of BiDi languages are Arabic, Farsi, and Hebrew. Those are the BiDi languages currently available in Catrobat and the ones we will focus on in this work. It may be useful to note that there are other BiDi languages, such as Urdu, Pashto, and Kurdish (which written with the Arabic script).

It is worth noting that the Arabic script is used for Farsi, as well as for Arabic. The difference being that Farsi uses extended letters. This can be viewed in the same way that English uses Latin script, whereas German uses an extended Latin script including extra letters, such as Ä, Ö, Ü, ß. This difference in language could influence layouts for text input whereas the UI should not be influenced. Concerning UI, Arabic, Farsi, and Hebrew render in similar ways. Small differences may exist, however, if nothing specific is mentioned, you can assume that all three languages have the same localizing solution. Note that this does not apply to text input.

Chapter 7

Visual Programming Environment for Children: Catrobat

In a visual programming language, the programmer arranges language elements on the screen in order to build a program, rather than by specifying them textually. Catrobat is a visual programming language developed for smartphones. Children can easily learn how to program without having to think about drawbacks like compile-time errors or complicated workows. Inspired by Scratch, Catrobat also defines blocks which can be snapped together in order to form a program. Unlike Scratch, Catrobat programs can be created and executed by entirely using smartphones.

7.1 Catrobat

Catrobat is a visual programming language inspired by Scratch. The main distinction between Scratch and Catrobat is, that Scratch needs a personal computer to run programs, whereas Catrobat programs are created and run by entirely using smartphones or tablets. There is also a version in development which can be run in any HTML5-compatible browser, be it on a desktop computer, or a smartphone, or a tablet [3].

Catrobat is a set of creativity tools for smartphones, tablets, and mobile browsers. Through Catrobat, computational thinking skills, as well as the free and open source software philosophy, are promoted in a fun and engaging way on a worldwide scale. Catrobat and the software developed by the Catrobat team are inspired by, but distinct from, the Scratch programming system developed by the Lifelong Kindergarten Group at the MIT Media Lab. Catrobat itself is an independent free and open source software project [3][72].

Catrobat provides an easy possibility for the children and teenagers to learn how to program without any programming knowledge, and enables them to program and wirelessly control Lego EV3, Phiro, and Arduino via Bluetooth [73]. In fact, Catrobat is created for children. Since it is created for children, it is very easy to learn and use. They can create animations. For older children or teenagers, they can create Catrobat games with

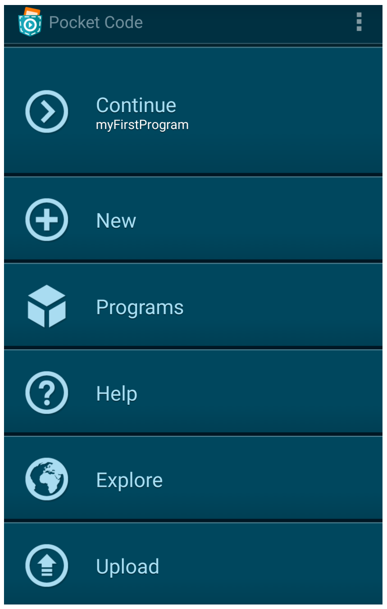
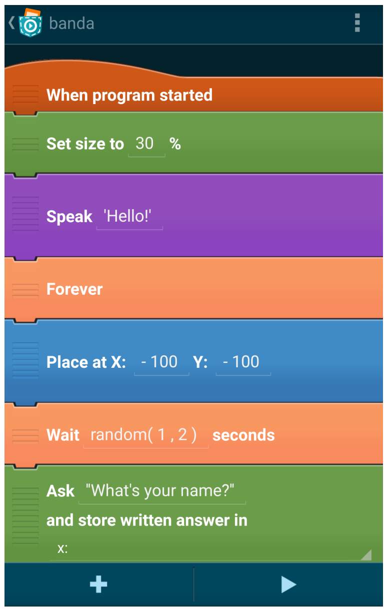


Figure 7.1: Screenshot for Pocket Code main menu.



When program started
Set size to 30 %
Speak 'Hello!'
Forever
Place at X: - 100 Y: - 100
Wait random(1 , 2) seconds
Ask "What's your name?" and store written answer in x:

Figure 7.2: Screenshot for Catroid program.

single-level or multi-levels. In addition, teachers and students in schools can use Catrobat to design effective education programs such as math quiz, physics simulation, and educational videos [72][4].

The version of Catrobat which is developed for Android smartphones is named Catroid and is available on Google's Play Store as "Pocket Code". Pocket Code allows children to create and execute Catrobat programs on Android, iOS, and Windows Phone 8 smartphones as well as on HTML5 capable mobile browsers. Catrobat's programs written on one platform can be directly run on all other platforms as well and can be shared via the Pocket Code sharing website. The Android version of Pocket Code currently works well on devices up to 7.0 inches. The iOS, Windows Phone, and HTML5 versions are still under development [69]. Figure 7.1 displays the main menu of Pocket Code.

Similar to Scratch, programs in Catroid are created by snapping together command blocks which are called "bricks" (see figure 7.2). The bricks are arranged in "scripts" which can run in parallel allowing concurrent execution. Just like in Scratch, Broadcast messages in Catrobat are used to ensure sequential execution of scripts.

7.2 Language Elements

The language elements are organized into the categories "Event", "Control", "Motion", "Sound", "Looks", "Pen", "Data", "Lego EV3", "Phiro", and "Arduino" as shown in Figure 7.3 . Each language element contains a set of bricks as shown below [74]:

Event: Elements of this category are used to sense events, which trigger scripts to execute.

Event's bricks are needful for every project without these bricks from this category, a project would not be able to start. By using bricks of this category broadcast messages can be sent and received, and events can be caught.

Control: Elements of this category are responsible for the program's flow, condition and loops can be defined.

Motion: Elements of this category are used to control an object's movement such as modify the position and the orientation of an on-screen object and can be used to set the gravity and mass for objects.

Sound: These elements play or stop a predefined sound file (sounds may also be recorded via the internal sound recorder) and alter the volume. Google's speak engine is used to read out some text.

Looks: The visual appearance of objects can be defined with these elements, the bricks are used to control an object's appearance. The size of an object can be set as well as transparency, brightness, and color. The elements of this category can be used to switch between the backgrounds. Also controlling some device peripherals is possible with elements of this category such as camera and flashlight.

Pen: The Pen category allows an object to draw shapes, plot colored pixels. The elements are used to change the size of the pen by a chosen number, set the size of the pen to a chosen number, and draw a copy of the object on the stage.

Data: Setting and changing the value of a variable is possible with elements of this category. The elements of this category also can be used to perform data manipulation operations over lists.

Lego EV3: Elements of this category are used to program Lego Mindstorms EV3 robot.

Phiro: Elements of this category can be used to program and control Phiro (educational robot) via Bluetooth.

Arduino: The elements of this category are used to program and control an Arduino via Bluetooth.

The visual elements are referred to as bricks. Bricks are arranged in virtual containers which are called scripts. A brick is always part of a script and cannot exist stand-alone, except for the script bricks "when program starts", "when tapped", "when you receive", "when screen is touched", "when physical collision with", and "when background changes to" which can form an empty script by themselves.

A script starts with one of the bricks "when program starts", "when tapped", "when you receive", "when screen is touched", "when physical collision with", or "when background changes to" followed by any other brick, except for another script brick, or nothing. Two or more scripts may follow sequentially. The list of scripts may also be empty. The elements "forever" and "repeat X times" generate loops and end with an "end of loop" brick. They contain zero or more other elements except for script bricks. They also may contain more "forever" and "repeat X times" bricks.

The element “if X is true then” introduces a conditional path of execution. The second path is started by the element “else”. The end of the conditional path is flagged by the element “end if”. There are zero or more other elements in between “if X is true then” and “else” as well as between “else” and “end if” except for script bricks. Those other elements may also be other “if X is true then”, “else”, and “end if” brick sequences.

The following list illustrates all brick elements of the Catrobat programming language (bricks which require a parameter are given with arbitrary values). Notice that the elements “when program starts”, “when tapped”, “when you receive ‘message 1’ ”, “when screen is touched”, “when physical collision with”, and “when background changes to” denote the start of scripts. The element “end of loop” cannot exist stand-alone, but is required by the elements “forever” and “repeat X times”. Similar, the elements “else” and “end if” cannot exist stand-alone, but are required by the element “if X is true then”.

Event

- When program starts
- When tapped
- When screen is touched
- When you receive ‘new message’
- Broadcast ‘new message’
- Broadcast and wait ‘new message’
- When 1 <2 becomes true
- When physical collision with ‘anything’
- When background changes to ‘New’

Control

- Wait 1 second
- Note ‘add comment here...’
- Forever
- If 1 <2 is true then... Else...
- If 1 <2 is true then
- Wait until 1 <2 is true
- Repeat 10 times
- Repeat until 1 <2 is true
- Continue scene ‘New’
- Start scene ‘New’
- Stop script/s ‘Stop this script’

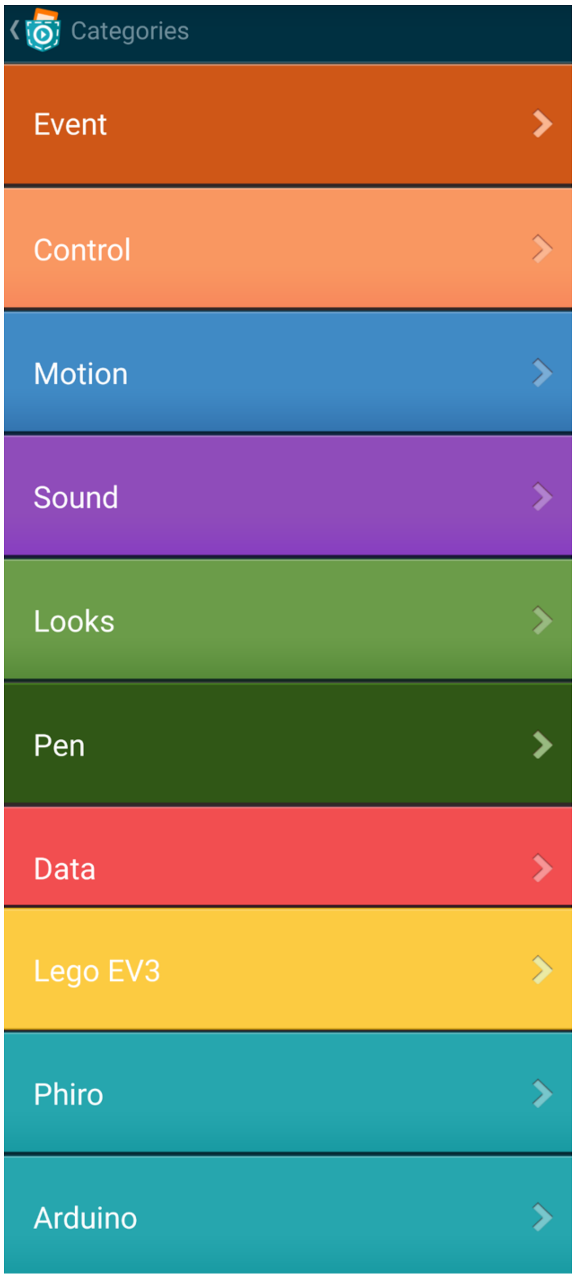


Figure 7.3: Screenshot for category menu.

- Create clone of ‘myself’
- Delete this clone
- When I start as a clone

Motion

- Place at X: 100 Y: 200
- Set X to 100
- Set Y to 200
- Change X by 10
- Change Y by 10
- Go to ‘Touch position’
- Move 10 steps
- Turn left 15 degrees
- Turn right 15 degrees
- Point in direction 90 degrees
- Point towards ‘New’
- Set rotation style ‘left-right only’
- Glide 1 second to X: 100 Y: 200
- Vibrate for 1 second
- Set motion type to ‘bouncing with gravity’
- Set velocity to X: 0.0 Y: 0.0 steps/second
- Rotate left 15.0 degrees/second
- Rotate right 15.0 degrees/second
- Set gravity for all objects to X: 0.0 Y: -10.0 steps/second^2
- Set mass to 1.0 kilogram
- Set bounce factor to 80.0 %
- Set friction to 20.0 %

Sound

- Start sound ‘New’
- Start sound and wait ‘New’
- Stop all sounds
- Set volume to 60 %

- Change volume by -10
- Speak ‘Hello!’
- Speak ‘Hello!’ and wait.
- Ask ‘What’s your name?’ and store spoken answer in ‘New’

Looks

- Switch to look ‘New’
- Next look
- Previous look
- Set size to 60 %
- Change size by 10
- Hide – show
- Ask ‘what’s your name?’ and store written answer in ‘New’
- Say ‘Hello!’
- Say ‘Hello!’ for 1 second
- Think ‘Hmmmm!’
- Think ‘Hmmmm!’ for 1 second
- Set transparency to 50 %
- Change transparency by 25
- Set brightness to 50 %
- Change brightness by 25
- Set color to 0.0
- Change color by 25.0
- Clear graphic effects
- Switch to look ‘New’
- Switch to look ‘New’ and wait
- Turn camera ‘on’
- Use camera ‘front’
- Turn flashlight ‘on’

Pen

- Pen down
- Pen up

- Set pen size to 4
- Set pen color to Red 0.0 Green 0.0 Blue 0.0
- Stamp
- Clear

Data

- Set variable 'New' to 1.0
- Change variable 'New' by 1.0
- Show variable 'New' at X: 100 Y: 200
- Hide variable 'New'
- Add 1.0 to list 'New'
- Delete item from list 'New' at position 1
- Insert 1.0 into list 'New' at position 1
- Replace item in list 'New' at position 1 with 1.0

Lego EV3

- Turn EV3 motor by 180 degrees
- Set EV3 motor to 100 % speed
- Stop EV3 motor
- Play EV3 tone for 1.0 seconds frequency 2 *100Hz volume 100 %
- Set EV3 LED status 'Green'

Phiro

- Move Phiro motor forward 'Left' Speed 100 %
- Move Phiro motor backward 'Left' speed 100 %
- Stop Phiro motor

Arduino

- Set Arduino digital pin 13 to 1
- Set Arduino PWM pin 3 to 255

7.3 Pocket Paint

Pocket Paint is a paint editor and image manipulation Android app which allows setting parts of pictures to transparent and zooming up to pixel level. It is integrated into Pocket Code, but it can also be used on its own. The Pocket Paint app has functions such as a stamp to copy-paste and resizes parts of images, a pipette to pick RGBa (red green blue alpha=transparency) colors, regular shapes (rectangle, circle/ellipse), or filling [75].

Pocket Paint is used for simple image editing such as rotating, flipping, zooming in/out, and cropping. The user can draw lines with different end shapes (circle or rectangle), and it also possible to thicken or thin the line by using the line width and style settings. In the Pocket Paint color chooser, you can choose colors either from a palette of predefined colors or through an RGBa dialog. Images are saved as PNG (with alpha channel) in the "Pocket Paint" folder and can be found via the Gallery and Photo apps. Pocket Paint is developed by the free and open source non-profit Catrobat project [4].

Chapter 8

Mobile Automation Test

In this chapter, we outline the basic testing activities, and briefly, explain the types of testing for mobile applications. Further, we discuss test equipment - real devices, simulators, and emulators. Each of them has benefits and drawbacks.

8.1 Testing Activities

As illustrated in some studies [76][77] mobile applications are not a bug and defect free software and the new software engineering approaches are essential to test that app. However, software testing is an activity which is aimed at evaluating program quality and also for improving it, by finding software's bugs and defects. Bearing in mind that the testing process and its results must be repeatable and independent of the mobile tester as well as consistent and unbiased [76].

Automated application testing means the automation of testing steps which are usually achieved by humans. Test automation is conducted using software tools referred to as test frameworks and it started before more than two and half decades since around 1990 [77] and can bring many benefits e.g., time and money saving and a better quality product which will be released earlier. Nowadays, any large closed (commercial) or open-source software product contains automated test suites to check its functionality. This special case for software products which are developed through many versions since automated testing is essential and beneficial in the case of regression and repetitive testing.

Automation is a solution that enhances testing efficiency, which is even more essential in mobile testing for many reasons. Since mobile testing must be conducted on many different fragmentations of devices, browsers, and operating systems, it is time-consuming, costly, and take a lot of effort to perform all the testing activities manually. Test automation can be used to decrease the time and costs correlated with testing. Moreover, test automation can be applied to enhance the productivity of the test team. On the other hand, automation testing is not designed to replace manual testing, it is to minimize the effort and time to market for the mobile apps.

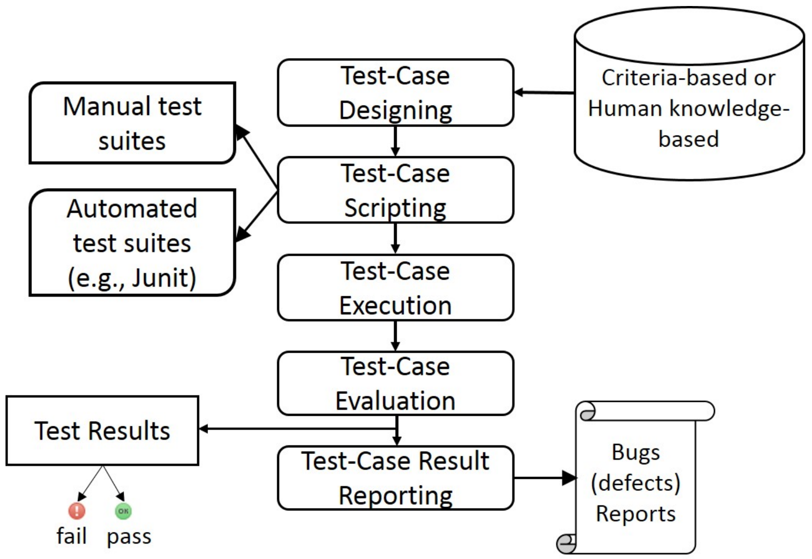


Figure 8.1: The five basic testing activities.

Typically, a testing process consists of several activities starting from test case planning to test case specification (test-case design), execution, and reporting [55][78], each of them can be either performed manually or automatically. To better realize how automation is implemented during the testing process (and not just during execution), based on numerous resources of test automation, five basic testing activities are introduced (see Figure 8.1) where a great potential for automation could be observed:

1. Test-case designing: nominate a list of test cases or test requirements to fulfill specific criteria, other developer' aims, or based on human experience.
2. Test-case scripting: write down or document test cases in manual test handwriting or automated test code.
3. Test-case execution: run test cases on the software under test and register the outcomes.
4. Test-case evaluation: evaluate the outcomes of testing (pass or fail).
5. Test-case result reporting: report test bugs and defects to developers.

8.2 Types of Testing for Mobile Applications

In this section of the chapter, we present testing types and how to test your app in different types and to make sure that it is robust, stable, usable, and bugs-free as possible. So, let us take a look at the types of testing that we should target for mobile apps. The following Figure 8.2 summarizes the essential mobile testing types.

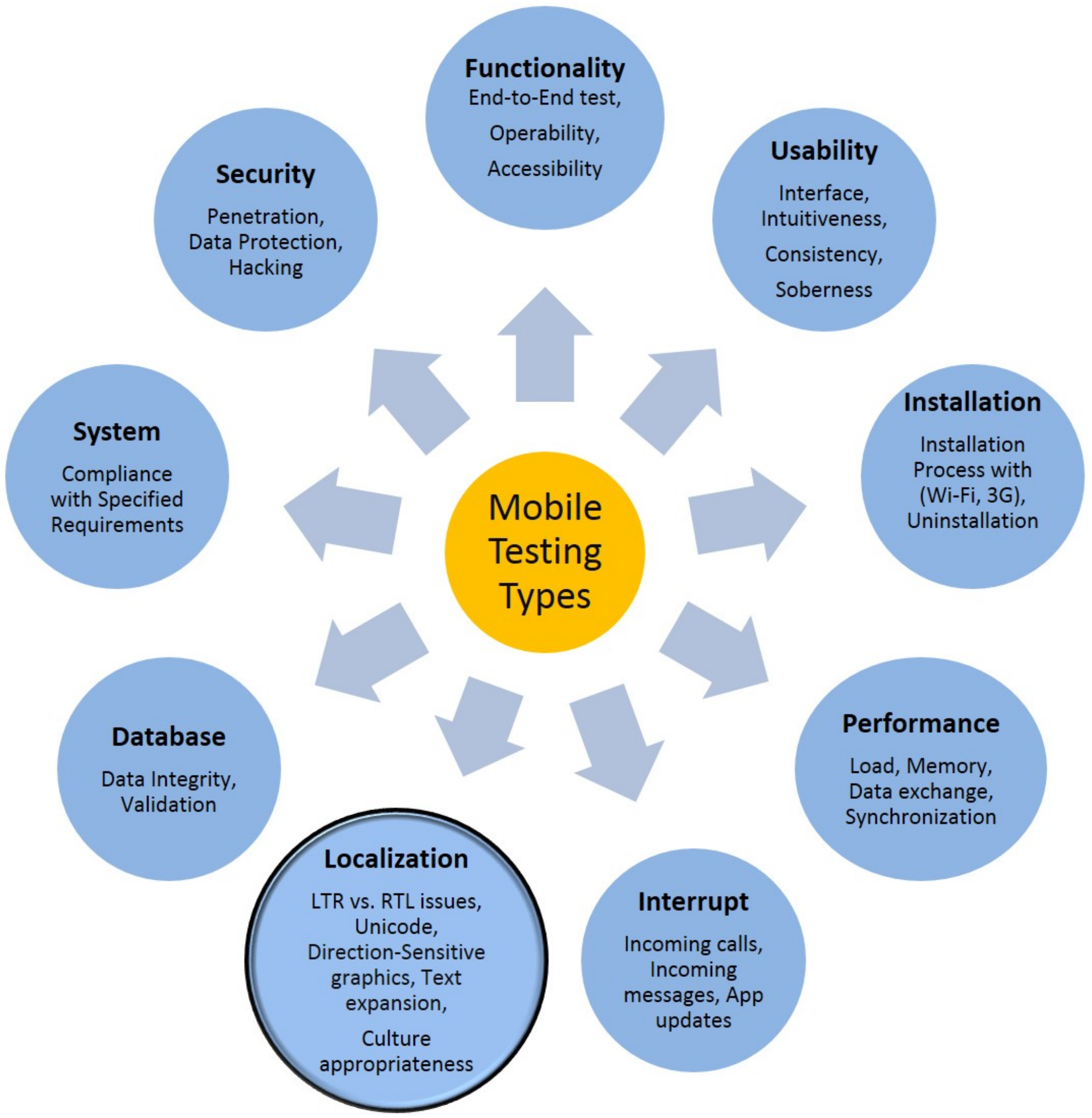


Figure 8.2: The types of mobile testing.

8.2.1 Functional Testing

The first thing that mobile tester has to do is functionality testing of the app, testing the functionality is one of the essential features of every software product. Functional testing ensures that the application features and requirements are implemented as they should be and to make sure all the functions correctly performed, for example, inputs, outputs, sliding, buttons, navigation, and data processing. Further, the functionality should be tested in different mobile user scenarios and environments, the sensors and interfaces of a mobile device should be considered while testing the functionality. Besides, the quality assurance criteria should be used to test the functionality of the app in a static and dynamic way [79][80]. If functional testing is performed on mobile devices manually, not automatically, it is going to be extremely difficult, exhaustive and time-consuming task due to different mobile-specific challenges.

8.2.2 Usability Testing

It is a good practice to conduct user experience testing very early in the test cycle. Most mobile testers plan to perform it in the last stage, but usability testing can reveal lots of defects with such UI elements as appearance, layout, content, fonts, graphics, text colors, background colors, etc. It is a good approach to find these bugs and fix them early in the cycle as possible. The domain of this testing depends on the guidelines of style, copy documents and content from the business requirements [80].

Usability testing is performed to verify whether the application is fulfilling its objectives and getting a favorable response from customers. The testing process ensures that the mobile app is now easy to use and provides a suitable user experience to the customers. This is essential since the usability of an application is the key to commercial success. However, the app should be easy to use; otherwise, it might be with low ratings, which lead to the app damaging and negatively affect the company's reputation. Efficient mobile usability needs lots of refining, intensive user research, and even more testing with real clients [75].

For instance, elements with the same type such as buttons or text views should have the same spaces, sizes, and colors. The mobile tester should inspect that all the UI elements are accessible, for example, that buttons can also be tapped by a user with thicker fingers and on different screen sizes and densities. The following list contains some aspect that should be checked during the usability testing like [75]:

- If the app supports multi-languages, the text should fit into every UI element and that the translation is correct.
- App shows friendly and useful error messages.
- App should offer undo and redo functions in order to provide an easy way to correct errors.
- App should follow the same workflow in every section.

- All UI elements such as buttons, labels, and other elements are big or small enough to be used.
- The navigation style of the app is easy to use.
- Text used within the app should be clear and easy to understand.
- UI elements must have the same look-and-feel, the same text, spaces, colors, and images.

8.2.3 Installation Testing

Mobile devices support two kinds of applications; the one which automatically comes with mobile OS, and another one you have to install from the App Store. The installation process is the first impression a mobile user can get about the app. Installation testing is a type of quality assurance work in the software industry that concentrates on what customers will need to perform to install and set up the new software successfully. The testing process makes sure that the installation, updating and uninstalling of an application are properly performed without any interruption. If the installation fails because of errors or defects, the user will neglect the app and never try to download it again. Installation testing verifies that the installation process goes smoothly without any difficulty.

To test the app for installation bugs, the following checklist should be performed [27]:

- Check that the app can be successfully installed on the local storage or memory card of the mobile.
- Check that the installation is done with various Internet connections such as Wi-Fi or mobile data networks.
- While the app is installing change the Internet connection (Wi-Fi to 4G, for example).
- Switch the mobile's Internet connection off, for example, to airplane mode, while the app is installing.
- Check the installation process when there is not enough space remaining in the local storage.
- Install the app by using the data cable or syncing from mobile-specific software applications.

Moreover, mobile testers should also test the uninstall process. So the app should be uninstalled and checked that it is completely deleted from the mobile phone with no data left on the hardware or local storage.

8.2.4 Security Testing

Security testing is a complex testing type that requires a lot of background in many different areas, such as client-server communication and software and hardware architectures.

It determines if an information system protects data and maintains functionality as expected. Security testing can be carried out on both client-side mobile applications and the server-side software to address all the vulnerabilities [80][81].

The following list provides the most common security problems of mobile apps [24]:

- Sensitive information such as passwords is cached on the device.
- Sensitive information such as passwords, tokens, or credit card details is accidentally stored.
- Sensitive information such as passwords is not encrypted on the mobile storage.
- Verification process for a password is accomplished only on the client side.
- Communication from the app to the backend systems is not encrypted.
- Apps use permissions for device aspects and peripherals that they do not need or use.

8.2.5 Performance Testing

Performance testing is one of the pivotal testing services in every software development project and especially for mobile apps. It determines how a system works in terms of responsiveness and stability under a specific workload. Mobile users expect an app to start/launch within two seconds; otherwise, they are unsatisfied and may uninstall it. Performance testing process is conducted to inspect the performance and behavior of the application under certain circumstances such as low battery, bad network connection, low memory, simultaneous requests to application's server by several users and other conditions. Furthermore, two sides might affect the performance of an application: application's server side and client's side [54][80].

Performance testing, a subset of performance engineering, is a computer science practice which strives to build performance standards into the implementation, design and architecture of a system. This type of testing decide what kind of performance is anticipated under such load, and evaluates the speed of response time for application under various network conditions (Wi-Fi speed, 4G connection etc.) Testers and developers can use performance tests to find out potential bottlenecks and deadlocks in their software application [24][81]. Normally, performance testing is carried out on servers or backend systems to inspect how the systems or software can handle huge numbers of requests and to match acceptable outputs for the users. The following list summarizes the simple mobile app performance tests:

- Compute the app's launch time.
- Compute the loading time of the content such as images, text, or animations.
- Check the delay time during operations or user interactions.
- Test on different architectures, especially on slower mobile phones.
- Compare the current app version with the new release.

8.2.6 Interrupt Testing

Interrupt testing is an important part of the mobile testing process. With the help of testing tools, mobile testers are able to define any potential performance or stability issues presented by an app. However, interrupt testing is a process of testing an application that functions may encounter interruptions like incoming calls, incoming messages or app updates. The different types of interruptions are:

- Incoming and outgoing SMS and MMS
- Incoming and outgoing calls
- Pushing notifications
- Media player on/off
- Volume buttons up/down
- Battery removal
- App updates
- Cable for data transfer insertion/removing
- Network outage and recovery

An application should be able to handle these interruptions by a transition into a suspended state and resume afterward.

8.2.7 Database Testing

Sometimes app's functionality requires local databases, in most cases an SQLite database, to keep data on the phone. Storing the data or the content of an app in a local database gives mobile apps ability to retrieve the content when the device is offline. The fact that dynamic mobile apps use databases means that mobile testers need to test them and the tasks that will be executed on the database.

This type of testing aims to check data integrity while adding, deleting, or modifying the data. In order to achieve good database testing, the mobile tester needs to get knowledge about the database model with the table names, procedures, triggers, and functions. Besides, database tools help the tester to connect to the device database for testing and verification objectives [27].

Mobile database testing based on the following kinds of tests [27]:

- Validation testing for database
- Integration testing for database
- Performance testing for database
- Procedural and functional testing
- Trigger testing

- Manipulation operations testing (create/read/update/delete)
- Database modifications testing to make sure they are shown correctly on the UI elements of the app
- Search and indexing function testing
- Security testing for database

8.2.8 System Testing

System testing of the application should check the overall compliance of the product with its specified requirements (functional and non-functional). It also tests a totally integrated system in order to verify its end-to-end functionality. Furthermore, it makes sure that the integrated system does not damage or corrupt its operating environment, or any other processes that the application communicates with. Concerning mobile applications, it is also important to test for functioning for each carrier. Due to simulators and emulators drawbacks which will be discussed in section 8.3.2, these tests should be achieved on the real devices in the real user environment in order to give the most realistic results (features of real hardware, targeted OS versions) [24].

In practices, system testing should be carried out in two phases. The first phase should utilize the test automation tools (Robotium and Espresso). This enables access to a large variety of devices on which the app can be simulated. The second phase is testing the application on real devices. This phase of testing supports the final decision regarding the release of the app.

8.2.9 Localization Testing

Nowadays, most of the apps are developed for global use and it is very necessary to care about regional trails like languages, cultures, time zones, etc. Localization testing is a type of quality assurance testing; it mainly focuses on the evaluation of the product's functionality, cosmetics, and quality of the localization. Moreover, it is a critical method that is carried out to control the quality of a product's localization for a specific geographical region or culture. This test is like a passport for your app to transfer from country to country [6][24]. Localization support has to be considered by the customer as one of the requirements in the initial stages of the application development lifecycle. It ensures that the localized software meets a local user's expectations in terms of language, features, and user experience in the traditional sense. Localization testing is usually performed on real devices and real user environment and is tested only on a handful of critical devices [6][16].

8.3 Test Equipment

During the last years, mobile phones have developed into robust, efficient, dynamic, connected and smart devices able to introduce all type of services at our fingertips. Everybody has a mobile phone and surfing the web app or installing the app on mobiles is becoming more and more customary. For this reason, the development of mobile applications is now seen a very gainful business in which customers are constantly in need of more high-quality applications. Based on this global situation, the business companies cannot bear to release an application that might have a critical defect or usability issue on the market [75].

If users do not have an excellent experience with the application, they will uninstall it and switch to a competitor product. For this reason, considering the necessary Quality Assurance (QA) on mobile systems has become an essential duty. But sometimes, this duty can be difficult since nowadays there is a large range of smart devices with a diversity of features. Because of this, the simulator has become an important equipment for development and QA teams. On the one hand, simulators are very vigorous equipment for developing mobile applications since they have distinctive features which allow them to provide a testing environment. On the other hand, because mobile applications are operated on real devices and not simulators, testing on real devices during the QA process is needed to assure the highest quality of an application.

Before developers start to test an app, there's one essential question to answer: Are they going to test it on a real device, in an emulator, or in a simulator? So, in this section, we will cover a part of the mobile test environment which includes test equipment-real devices, emulators, and simulators. This section is based on [27][82][83] and my personal experiences in the mentioned equipment.

8.3.1 Real Devices

Testing on real devices is necessary for every mobile application since it offers many of advantages as compared to an emulator/simulator. The tests will be executed in a real user environment and therefore it gives the most realistic results. However, testing on emulators will never replace testing on real devices. The mobile tester can accomplish all required test cases on real devices which offer the possibility of using the full device's hardware, software, and device-specific features such as sensors, CPU, GPU, and memory. On the other hand, it is far from easy and costly to test application on all available real devices.

Real devices benefits

- Reliability - because developers ensure that end user with same device description and with same configurations will not have any problem or defects.
- Use the features of real hardware and software - if tested application interacts with sensors, camera, Bluetooth, or NFC, then tester has to turn on a device in his hands. Some emulators imitate behaviors of these features but their results are not very closer to a real user experience.
- Real user experience - to obtain a trusted user testing especially in performance, it is necessary to handle a real device. Mainly, the application will be run by the different

type of customers and on different real devices, and testing on real devices always gives us actual user experience, because it is the only way to correctly realized the user experience. Moreover, performing testing on real devices always provides us with accurate results and we will locate actual application defects.

- Interactions with real data networks - test cases are achieved in real devices with real data networks and the result is close to the real user experience.
- Interoperability testing - some of the mobile applications will connect to mobile communications networks. It is a significant task to achieve interoperability testing in a live network with the real device because if we achieve this type of testing using a simulator, the connection of network will be different. With testing on real devices, we can expose jitters, because of interference and crosstalk with other signals which can influence the performance of processors in a mobile application.
- Performance testing - is always a critical type when testing. This type of testing is difficult to be performed with a simulator, but, when using real devices, it is easier to find performance issues, as well as issues which are the result of the device itself or its environment. Further, we will expose crashes and memory leak problems which cannot be exposed on an emulator.
- Security testing - a widespread issue amongst mobile users is security. Users are sentient to private data, like bank account numbers that kept or saved on devices, or if passwords are shown on the screen. Testing for these kinds of security issues in a simulator or emulator is not a good or beneficial testing since it is necessary to test the behavior of the real devices. In addition, one of the challenging aspects of security testing is not all real devices have identical security designs, therefore each device must be individually checked.

Real devices have not been created for testing objectives. It means there are some drawbacks to consider when the testing processes are conducted with a real device.

Real devices drawbacks

Testing on real devices also has some drawbacks. The following points summarize the disadvantages of testing using real devices:

- Variety of devices/costs - developer want to verify that the app runs on all the new features (hardware and software), therefore, testing on few real devices does not guarantee that the application will work on other devices, as a result buying new test devices is very expensive. For instance, if the development team has an application that targets a plenty of real devices, all with different hardware features, specifications, and service providers. It is infeasible to obtain all the needed mobile phones to test on. But if the enterprise could obtain all the needed mobile phones, it will take a lot of effort to accomplish the testing processes in all the real devices.
- Limitations of mobile carrier - it is not easy to test how mobile application will behave on the different mobile carrier because different network infrastructures are available.

- Maintenance of all devices - the maintenance process is time-consuming because developers need to keep all devices up-to-date to newer versions.

8.3.2 Emulator and Simulators

The Android emulator is a desktop application that interprets the instructions of the compiled app and behaves like a real mobile device running an Android operating system. It enables developers and testers to run Android applications. Not all of the mobile device-specific hardware features such as sensors, Wi-Fi, or touchscreen gestures can be emulated. Emulation makes it possible to gain information about the behavior of the app on different Android devices, e.g., with different screen resolutions and operating system versions which are a vital feature for continuous integration and parallel testing without the need for physical devices.

Simulator mimics all of the typical hardware and software aspects of a typical real mobile device environment. Simply, the emulator is an imitation of a real device. Simulators, like iOS simulator, are less complex software applications that imitate a small part of the device's behavior and hardware resources like CPU, memory, network, etc. Primarily, simulators are identical to the target platform and this makes them much faster than emulators. Therefore, the testing results of simulations could be biased. With simulators, it is not feasible to test device-specific hardware features.

By the way, simulators are helpful at the initial phase of the development process in order to handle feedback about the implemented features. The simulator provides some features that help us to start testing steps. These are the critical advantages to achieve testing cases on a simulator. For instance, learning to fly a plane using a simulator is much safer and cheaper than learning on a real plane.

The biggest difference between a simulator and an emulator is that a simulator aims to replicate the behavior of the mobile device, while an emulator attempts to replicate the entire inner architecture of the mobile device more precisely and therefore it provides more realistic results.

Developers can use emulators and simulators only to test a simple functionality or ensure that the application's look-and-feel is acceptable. Mobile platform companies offer either a simulator or an emulator. Apple provides a simulator; while Google and Microsoft provide an emulator. In practice, all existing simulators are unable to simulate real-world smartphones with their sensors, GPS, connectivity.

Emulator/simulator benefits

- Costs - both emulators and simulators are free to use and are delivered with the SDKs from the different mobile platforms. Mobile phone vendors have made huge efforts to make sure that their platforms are simple to test and that there is a great range of solutions available. These tools with a very high quality, they are very reliable and there is a lot of diversity.

- Up-to-date version - a new version of an emulator is always enclosed with a new version of SDK.
- Simple to use and configure - application can be easily distributed and installed from Integrated Development Environment (IDE) and developer can configure different options such as operating system versions, screen size and resolutions, location, connection type, camera, accelerometer, etc.
- Simulators and emulators are important friends for development and QA teams. They save a lot of time and money, but obviously, there are some drawbacks if we are achieving testing activities on a simulator or emulator.

Emulator/simulator drawbacks

- Non-Realistic test results - using emulators/simulators increases the risk of ignoring great bugs that discovered only on real devices since emulators/simulators are not the same as the real environment and there is always a possibility that the application could act slightly different on a real device because of different hardware and software features.
- Limited performed test cases - emulators/simulators fail to provide a real device environment with all of the relevant sensors and features. For instance, if the application implements NFC or compass, it would be very difficult to test the application that interrupted with incoming calls or SMS.
- Not real data networks - network speeds can be simulated, but this will not include the real data networks with traffic loss or change in network speeds and technologies.
- Different performance for CPU and GPU - emulators/simulators do not present the same performance as real devices since they do not reflect the actual hardware and software's features of each real device.
- Low responsivity - some emulators react slowly especially in applications that implement animations or games. Consequently, the test results are delayed. Further, depending on the processing power of the computer which is running the emulator, with low CPU and memory, performance on the emulator may be unrealistically low or high.

We have seen both benefits and drawbacks of testing on the emulator, simulator, and testing on real devices. Indeed, emulator and simulator are great, but we should not presume that just because our app runs completely on them, it will run in the same way on the real handset. However, sometimes there is just no replacement for the real thing. For this reason, emulator/simulator are a very beneficial step in any testing process, but should never be used to replace real device testing, since it provides you always a real situation of your application. A perfect approach is to use emulator/simulator during the initial development and testing stages as a prerequisite for testing on real devices when the ripeness level of the application is low. Hence, simulators will speed up the detection and fixing of bugs, and the use of real devices will be in the last stages.

For all that, the decision at which stage a simulator or a real device should be used depends on the enterprise. Many factors should be carefully considered, as due dates, financial support, personal involved in QA steps and user needs. To perform effective testing for mobile web and apps, a combination of emulators, simulators, and real handsets is required. This has to be a mixed approach of emulators, simulators and real handsets. Using the combination of an emulator/simulator and the real device, along with a robust test process, brings us to the point where our product is stable and sounds to be running perfectly on a huge number of real devices.

Chapter 9

Background and Related Works

There are two stages of globalization testing: internationalization testing, and testing for localization. In this chapter, we provide the theoretical background of internationalization and localization testing and discuss some studies related to such types of testing.

9.1 Internationalization Testing

Internationalization testing aims to detect potential bugs in software globalization. The process makes sure that the code can handle all international support without breaking international functionality issues that might cause either data loss or rendering defects. It aims to reveal the issues of international functionality prior to the products' global release through product testing. This type of testing checks whether the product will be correctly adapted to work under several languages and regional settings (ability to render linguistic characters, to run on non-English operating systems, to display the correct date and time formats and numbering systems, etc.). The goal of internationalization testing is to test a product which is supposed to work uniformly across multiple locales and cultures. In addition, the testing process is designed to prove that a product is capable of functioning in a globalized mode, and is independent of any certain localization.

Internationalization testing does not form a test phase by its own, but rather should be part of functional testing. These are some of the areas it should cover:

- Unicode support and non-Unicode support (Chinese and other Asian-language character sets)
- Double byte character set support
- Bi-directional languages support (such as Arabic or Hebrew)
- Date and time formatting
- Locale specific settings and formatting (number formatting, currency formatting, units of measurement, etc.)

- Input method editors (IME) input support

Benefits of internationalization testing:

1. Increase in international visibility to the product
2. Increase in product downloads and revenues
3. Increase in product quality, if it is developed to cover internationalization issues

9.1.1 Is Automation Possible?

Internationalization testing is always a challenge for the mobile applications vendors. Each language that the application may support, could increase the number of test cases needed for testing. If the vendor uses automation extensively, the automation testing scripts should be designed and maintained for an app. Therefore, to cover this issue, the mobile tester really needs to design test scripts that could be easily extended to support several languages.

As discussed above, if the application is created using the activities of internationalization and localization to release a language-neutral version, in which configuration files, are used to cover language peculiarities. The application reads from the respective resource files at runtime to present it in the chosen language. In this approach, the basic version of an application is only one, which has the ability to switch the language of the user interface when it is needed. In practice, it is possible to automate the internationalization testing since such test cases are created only in the base language and easily extended for all languages.

Before starting with automation testing, we need to make sure that the names or IDs assigned to all the elements in the activities and dialog boxes do not change with the change in the language. All the views on the application such as text field, button, text view, check box, radio button, toggle button, spinner should have a unique id (like name or ID) and should be independent of language. If a mobile tester is satisfied that all the layout elements are not language-dependent, then he can build efficient automated internationalization testing methods for the target application.

9.2 Localization Testing

Localization testing controls the quality of a product's localization for a destination locale or culture [6]. Localization testing exposes deficiencies of the software or application in design, linguistics or the overall UX, to be fixed before launching into the market. However, the user interface, documentation, and content can be in different languages, calendar, date formats, and units of measurement. With such complexities, mobile vendors need to ensure that their applications are relevant to the locales they serve. Internationalization and localization testing ensure reliability, usability, and acceptability of the product to the users worldwide. Localization testing should cover the following main categories:

Linguistic testing should cover the following issues:

- Verify that the translation is translated in context (i.e. accuracy of translation).
- Verify correct grammar.
- Verify correct spelling and no punctuation errors.
- Verify adherence to the glossary.

Cosmetic testing should cover the following issues:

- Verify consistent layout with the original product.
- Verify consistent layout with the target culture.
- Verify translation consistency.
- Verify correct line breaks.
- Verify truncation issues.
- Verify that the UI elements are not overlapped.
- Verify that the layout is flexible.
- Verify using of culture-neutral icons and logo.

Functional testing addresses the following issues:

- Verify the correct render for all special characters.
- Verify correct date and time formats.
- Functional validation of localized product: install, use, uninstall and supported platforms.
- Verify that all the elements work properly (buttons, hyperlinks, saving, IME, etc.).
- Verify that the hot keys and shortcut keys work properly in the localized product.

9.2.1 Automation Localization Testing

One way to introduce efficient localization testing and less error prone to human is to have it automated. This is true for any software testing type and can be applied for localization testing as well. A fully localized product will take a lot of time and resources to verify all UI elements issues, characters rendering, and localized resources manually. It is especially time-consuming, if the testing requirement is to compare all functionality to the source product (usually the English version). Using localization testing approach is one solution. If the automated testing solution itself is internationalized and each locale's resources created, then the same automated tests can be executed for all localized versions.

9.3 Related Works

Most of the previous research activities with similar intentions of testing internationalization and localization issues have been considering mainly the checking of desktop and traditional web applications, and most of them focusing, in particular, on an adaptation of time, date, number, and currency formats rather than the GUI issues and User Experience (UX).

In [13]. Kopsch reviewed many localization issues such as:

- Requirements of hardware and software standards, in particular, Kopsch considered the different external input devices, different monitors, and different components, and what is the relationship between mobile devices and desktop computers? And which operating systems, browsers or third-party components are used in the local region.
- Character set and special characters: Especially Chinese character sets should be recognized and the specific special characters would affect the target application.
- Number formats: What additional number formats are required for the system? Of significant concern are different formats of dates, times, measures, weights, and numbers (e.g., decimal places), or zip codes and telephone numbers. Further, Kopsch also considered the additional input fields that might be required. Primarily the target software is concerned with the formats of dates. At different points in software.
- Graphical User Interface: For localization, the translations of the buttons, labels, navigations, and menus would be necessary. This poses the question as to whether the translated texts from one language to another would change the length of the string for the existing layouts and buttons. What styling bugs should be expected by software localizers? And would these bugs then require creating some new styling?
- Different output formats: An important feature in the target application is the ability to export different doc types into different formats. What document standards are there in the new locale? And if we need to create new output formats or not?

Furthermore, Kopsch discussed the necessity of the better communication between the project team, and the customers in order to finish all the customers' requirements which were incompletely specified and avoid any ambiguity or miscommunication.

In [14], Cavalleri mentioned that when developers are testing localization, they do not just have to verify whether the translation is correct in linguistic terms, they also need to consider cultural issues. Even if the localization testing is restricted to the language field, testers should not only verify that the translation is correct from the source language to the target language, they should also consider the particular characteristics of each destination. As a result of the translation, there are also some cosmetic checks that need to be implemented. For example, one needs to check that the labels still fit on the screen, whether they were clipped, that they are not overlapped with other UI elements, and so forth as.

Cavalleri pointed out that the defects, which software testers usually find in localization testing, are related to text expansion, truncated strings, overlap of GUI elements, misalignment, date and time format, corrupt characters, untranslated images, shortcut assignment (i.e. duplicated, missing), hard-coded strings (i.e. untranslated) and unsupported code pages. The application may also behave differently in different countries, so you must not dismiss the functional testing of these business rules. Other adaptations you must also bear in mind are the different regulations and laws that restrict the system in that particular country. For instance, the correct use of the local currency and date formats.

Cavalleri focused on considering the colors meaning during localization phase. However, sometimes the colors are used to convey information. For example, we usually alert the users with a warning in red. By doing localization testing, we must keep in mind that colors do not have the same meaning in every culture. For example, the colors white and green. While in most countries the white color means purity and is even the color chosen for marriage, in China it often means death. Moreover, in Wall Street, Americans use the green color to show increases in share prices, while in Asia, green shows a drop in prices.

Cavalleri sees that localization testing is not restricted to verifying the translation since it should cover many other things to test. However, a tester may only want to check the translation of the application into different languages. If the number of languages is large, these tests are candidates for automation. The major advantage of automation in these cases is that tester can use a single test script for each scenario. This script is written once and run many times as needed according to the languages under test.

During the localization process, many good practices and tips should be considered, and the tester should follow the localization checklist. Here is a simple example of a checklist suggested by Kotze in [19]:

Internationalization phase

- Formats for dates, times, addresses, and telephone numbers
- International paper sizes
- Units of measurement and currencies
- Display text and fonts correctly
- Do not use language to assume a user's location, and do not use location to assume a user's language
- Colloquialisms, Jargons, and metaphors
- Technical term, abbreviations, or acronyms
- Icons or images that might be offensive
- Political offense in maps or when referring to regions should be avoided
- Sorting

Localization phase

- Localizable resource files

- Resources of app that need localization
- Text size
- Right-to-left direction
- Image and audio files for localization
- Text in an entire sentence
- Text in different contexts

Kotze noticed that the software testers should make it a priority for them to reveal I18n and L10n defects early in the development phase by concentrating first on five languages including English. Experience has shown that we are most likely to find specific issues in the following languages:

- German: It contains long words that can expose dialog size and alignment issues better than other languages.
- Japanese: With tens of thousands of characters, multiple non-Latin scripts, alternative input method engines, and an especially complex orthography, Japanese is a significant way to find defects that affect many East Asian languages.
- Arabic: It is written right-to-left and has contextual shaping where a character shape depends on its position in a word and the surrounding characters.
- Hindi: This will help to find legacy, non-Unicode defects that affect all such languages.

Finally, the tester should make sure that the defect tracking system which the team uses handle all the supported languages. He also recommended testers to use a single management system for all supported languages as this will streamline and enhance the reporting efficiency. Further, testers should manage multilingual defect reports using automatic approach by examining individual defects and then automatically translate one-by-one for appropriate follow-up by the feature team that owns the affected component.

Testers can build a language detection API for the defect tracking feature that is able to automatically detect the language of the customer defects as they are reported. Or just have an option available for the reporter to choose the language. If the defect tracking feature has the ability to forward raised defects according to a particular value in a specific field to a specific set or individual, then this could be helpful as well.

In [16] Abufardeh and Magel proposed testing methods that can be employed to test significant issues of bi-directional software in general and specifically Arabic software. However, these testing methods are discussed in the context of the Dynamic Systems Development Methodology (DSDM) process lifecycle that merges internationalization and localization activities into the software lifecycle. Abufardeh and Magel outlined the critical aspects of bi-directional languages in bidirectional software which includes the following:

User interface elements: User interface elements must be created to accommodate the functional and cultural specifications of the end-user. Internationalization purposes are the neutralization of UI elements and the elimination of language and culture-specific properties of the UI elements in preparation for later localization. These elements include the following: Windows, dialog boxes, message boxes, menus, toolbars, text boxes, help balloon, images, icons, colors, and sounds. Further, when dealing with bidirectional languages, special attention should be given to direction sensitive graphics, such as icons that have a specific directional orientation (back, next, undo, and redo) since navigation controls should be consistent with the right-to-left text reading order and with right-to-left UI layout.

Text directionality: However, the directionality of text becomes an issue when dealing with Arabic, Persian, Urdu, and Hebrew texts, which are read and written from right-to-left. For these specific scripts, the logical order (the order in which the user enters text with a sequence of virtual-key inputs) and the visual order (the order in which characters are represented to the user) are different in most cases.

Input method editors (IME): With IME, a localized version for the keyboard is not necessary for the users to enter ideographic characters. For example, a user of a Western keyboard can input Chinese, Japanese, Korean and Indic characters. The key issue is that none of the element's string or culture-specific components should be hard-coded in the source code. In other words,"All this string should be moved into resource files" but still accessible by the software. Usually, the content of these resource files can be retrieved by the software at runtime to supply these elements to the users in their local language.

Language and UI layout: Language's scripts and screen layout of a user interface are closely correlated to each other. Different languages are read and written in different directions. For example, bidirectional languages like Arabic and Hebrew are read and written from right to left. Further, numbers are read from left to right with the highest order digit on the left side. Therefore, the UI layout should be mirrored when software is localized to the bidirectional languages.

For the localization testing, the proposed approach must be done at both the individual UI element and the screen level. At the individual element level, the following tests are done:

1. The UI element is doubled in size.
2. The UI element is halved in size (from the original size).
3. The color of the UI element is changed.
4. The UI element is rotated ninety degrees counterclockwise to show a change in the shape of the elements.
5. If the UI element contains text, one line elements are adjusted to multiple lines and multiple line elements are adjusted to two lines.

In each case, the full set of the tests applied to the original element is applied to the customized one. These tests might include selecting the element, unselecting the element, typing text into the element, altering the contents of the element, and so on. The final part of full development testing checks the screen and other user interface units larger than a single element (for example, menus).

Testing methods for bidirectional software:
Abufardeh and Magel provided essential, full development, and user acceptance testing methods for a few of the globalization issues of bi-directional software. The efficient testing methods are proposed to check the following issues.

1. Text directionality: Text directionality of all user interface elements must be language-neutral. Testing must be performed at two levels. Each individual element must be tested or each label, text field, button, screen or menu. Further, the facility that supports localization of the elements must be tested. Important testing could be performed by supplying each user element with a variety of strings from the targeted languages using the localization facility. In this test, each element must be supplied with a variety of strings from various languages. This testing should result in a diversity of changes in the text directionality and content format for that element.

   Another result will be different elements showing different strings from different languages. Such testing should need minimal effort and thought by the developers. Full testing must be performed at both the individual element and the screen level. At the individual element level, the following tests should be performed:

   (a) The UI element is supplied with mixed-language strings from languages with similar fonts.

   (b) The UI element is supplied with mixed-language strings from different languages with different fonts.

   Another distinction of this test is that tests (a) and (b) are done with various string lengths. Also, another test distinction is that tests (a) and (b) are done with single and multiple text strings. In each case, the full set of the tests applied to the original element is applied to the customized one. These tests might cover selecting the element, unselecting the element, typing text into the element, changing the contents of the element, and so on.

2. IME: The issue of an IME is not completely a user interface issue; rather, it is a keyboard issue. The potential need to handle text input characters and symbols that do not exist on the keyboard is not an insignificant issue. Essential testing for IMEs is done by using a localization facility that simulates a keyboard keystroke to supply UI elements with data, such as text field or label. The input is then saved into flat files or database tables, and then it is displayed back into the UI elements.

3. User interface layout: Unlike the previous issues, the issue of UI layout addresses windows, messages, dialog boxes, menus, toolbars, status bars, graphics, and tables and it is an essence of user interface issue. The potential need to support left-to-right and right-to-left UI layouts is considered substantial in bidirectional software that demands mirroring.

   Essential testing for UI layout is done by using a localization facility that transforms the coordinate of one, a set of UI elements or full screen from left-to-right and right-to-left, i.e. mirroring. The logical relationship and the directional attribute should be maintained before and after mirroring. Full testing must be performed at the individual element level, on a set of elements, and on the screen as a whole. At the individual element level, the following tests are performed:

(a) The coordinates of the element are transformed from left-to-right.

(b) The size of the element is increased by 1/3 or 1/2 (from the original size).

(c) The height of the element is increased by 1/3 or 1/2 (from the original height).

(d) If the element contains text, one line elements are adjusted to multiple lines and multiple line elements are adjusted to two lines.

These tests might include selecting the element, unselecting the element, typing text into the element, changing the contents of the element, and so on.

Chapter 10

Bidirectional Localization Testing Architecture

Automated testing helps developers control the progress of system development, and to find defects early and efficiently. This chapter presents an overview of the tools which are used by the proposed automated bidirectional languages localization testing approach such as Android tools, Robotium and Espresso testing frameworks, continuous integration system (i.e. Jenkins), Git, and Gradle.

10.1 Android Tools

The Android Instrumentation Framework from Google is the efficient base for most of the Android test automation tools. If you want to start with Android test automation, you need to understand the view hierarchy of an Android app. Further, getting knowledge about what types of components and elements the app implements is necessary and how all these elements are laying out on the screen. UI Automator Viewer is a useful tool provided by Google, the functionality of this tool is to check the app's view and layout hierarchy. This tool enables developers to view the properties - the name or ID of each UI element that is shown on the screen. This type of information like (name or ID of an element) is very necessary for developers to write the mobile test cases [27].

10.2 Robotium Framework

Robotium[1] is an open source test framework that is used to create a robust and powerful gray box for Android applications. It provides full supports testing for native and hybrid applications. Robotium also provides a simple and graceful API used to write UI test cases [27]. The framework can be used both for testing mobile apps where the source

[1]https://github.com/robotiumtech/robotium

code is accessible and apps where only the APK file is accessible and the implementation specifications are not known.

Robotium was started in January 2010 by Renas Reda. He is the originator and main developer for Robotium. The project initially developed with v1.0 and continues to be with many releases due to new requirements. It provides a full support for Android features such as activities, toasts, menus, context menus, web views, and remote controls [27][84].

Robotium framework is released under the Apache License 2.0. It is a free testing framework and available to be downloaded from the Internet. It can be easily utilized by individuals and enterprises and is built on Java and JUnit 3. Robotium is also considered as an extension of Android Instrumentation Framework. Further, it provides the Solo object to call methods such as clickOnText or enterText [84].

Robotium provides an excellent readability benefit, so when the developers read the test cases, they can get knowledge about the scenario of test cases and what will be tested on the UI views of the app. It requires only minimal knowledge of the app under test. However, Robotium tests are written in the Java programming language and can either be executed on the Android Emulator (Android Virtual Device (AVD)), or on the real device.

Currently, it is the world's leading Automation Testing Framework, and many developers are contributing to building more and more usefulness features in future releases. Since Robotium is an open source project, any developer can introduce his contribution for the target of developing and enhancing the framework with many more features [84].

10.2.1 Features and benefits

The following are some features and benefits of Robotium [84]:

- Enables developers to write a reliable test case in a quick manner with minimal knowledge of the application that developer wants to test.
- Offers APIs to directly connect with UI elements within the Android application such as text view, edit text, and button.
- Provides methods to capture screenshots.
- Can be used without a source code.
- Supports Android features such as activities, menu, and context menu.
- Its framework can handle multiple activities in an Android application.
- Can inspect the messages that are displayed on the screen (toasts).
- Supports scrolling of app activities.
- Robotium test methods can also be executed using the command prompt.
- Robotium automated tests can be quickly written.

- Its Recorder enables developers and testers to record Robotium test cases instead of writing the code manually.

10.2.2 Robotium Extensions

An exciting library which extends Robotium is developed by the company Bitbar; however, it is called the ExtSolo. The ExtSolo library provides some very useful test methods to Robotium such as the following:

- changeDeviceLanguage(java.util.Locale locale): Enables developers to change the current language of the device during the test execution.
- setGPSMockLocation(double latitude, double longitude, double altitude): Enables developers to set the current GPS location for the device.
- turnWifi(boolean enabled): Enables developers to turn Wi-Fi off and on.

10.3 Espresso Testing Framework

Espresso[2] is a testing framework created by Google Inc. Espresso is a very powerful UI testing tool for Android application. It was released by Google to enhance GUI testing features for Android applications and to make it easy to create reliable UI test cases, the objective is to introduce a simple API with minimal lines of code. Espresso is automatic testing for a user interface. Once mobile tester writes the test case code, Espresso will execute using events: do several things that a normal user will do, then verify the test (unlike JUnit testing where there are no UI aspects). Espresso allows developers to easily write efficient user interface test cases for a single application.

Google's Espresso testing framework offers a set of APIs to write UI tests scripts to test user's actions and behaviors within an app. These APIs enable a mobile tester to build automated, reliable, and concise UI tests. The framework is well-suited for creating white box-style automated test cases, where the test cases utilize implementation code details from the app under test [85].

Nowadays, Espresso is a part of the Android Support Repository. It simulates all user events and interactions with a smartphone (typing, clicking, swiping etc.). In addition, the framework automatically synchronizes test actions with the application UI. Espresso will keep an eye on the UI thread (the thread that is responsible for drawing all the pixels on a screen). Before Espresso, an ActivityInstrumentationTestCase2 was used, the test case will be failed since the view was not drawn yet and the test case was trying to click or type on it. In Robotuim, mobile testers use some Sleep statements hoping that after sleeping the pixels will be drawn correctly and then they can try to click or type on it. Since it is based on automatic synchronization concept Espresso will loop the Sleep until the UI thread is ready then it will verify and do some actions. Therefore, with Espresso mobile tester do not need to call Sleep methods [85].

[2]https://code.google.com/p/android-test-kit/wiki/Espresso

Espresso exposes a simple API to connect directly with the UI elements of a native Android app. The framework also makes sure that application's activity is launched before the tests start. It also enables the test case wait until all observed background activities have finished. If Espresso is used for testing outside Android app, the mobile tester can only perform black box testing, because a tester cannot access the classes and libraries outside of an application.

The key features of the Espresso testing framework include:

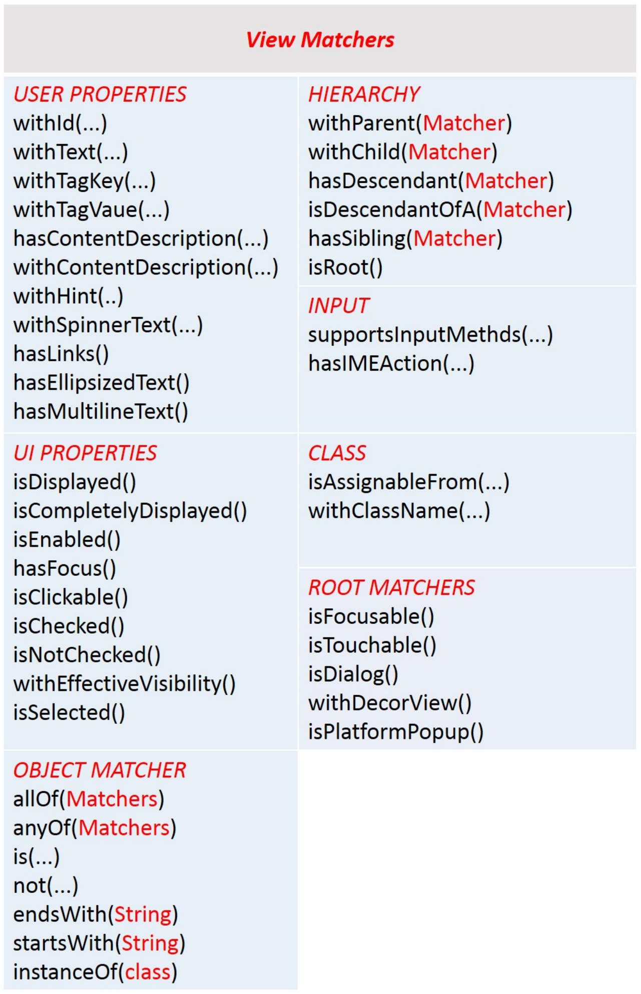
- Espresso code resemble English, which makes it foreseeable and easy to learn
- Flexible APIs for a view and adapter matching in target apps. For more information, see ViewMatchers
- An extensive set of action APIs to automate UI interactions. For more information, see Action APIs
- UI thread synchronization to improve test reliability
- Espresso API is comparatively small, and it is open for customization and improving
- Espresso tests run much faster (no waits, sleeps)
- Android studio version and Gradle support

10.3.1 Espresso API

The Espresso API encourages mobile tester to consider in terms of what a user might do while running the application - locating and interacting with UI elements. At the same time, Espresso limits direct access to activities and views of the application since holding on to these elements and acting on them off the UI thread is the main cause of test flakiness. Therefore, a mobile tester will not see methods like getView and getCurrentActivity in the Espresso API. The Espresso API is very simple and well structured. It is written in Java language and the framework is based on the Android instrumentation framework. The mobile tester can safely work on views by implementing Espresso ViewActions and ViewAssertions [85].

Espresso is an extensible testing framework, it introduces flexible APIs for views and adapter matching, so mobile tester can create his own matchers in target Android app. Furthermore, Espresso introduces a simple API, automatic synchronization of test cases actions with the user interface of the tested Android app. However, JUnit4 supports, and rich information of failure make it a great tool for UI testing. In the field of testing, Espresso framework is considered to be a complete replacement for Robotium framework.

However, Espresso supports test execution either from the command line, from an IDE, or from a server of continuous integration on real devices or emulators [27]. In comparison with other Android test automation tools, the execution of test case written using Espresso is much faster since no need to waitFor() and sleep () methods. Basically, Espresso has four main components [85]:



View Matchers

USER PROPERTIES
withId(...)
withText(...)
withTagKey(...)
withTagVaue(...)
hasContentDescription(...)
withContentDescription(...)
withHint(..)
withSpinnerText(...)
hasLinks()
hasEllipsizedText()
hasMultilineText()

HIERARCHY
withParent(Matcher)
withChild(Matcher)
hasDescendant(Matcher)
isDescendantOfA(Matcher)
hasSibling(Matcher)
isRoot()

INPUT
supportsInputMethds(...)
hasIMEAction(...)

UI PROPERTIES
isDisplayed()
isCompletelyDisplayed()
isEnabled()
hasFocus()
isClickable()
isChecked()
isNotChecked()
withEffectiveVisibility()
isSelected()

CLASS
isAssignableFrom(...)
withClassName(...)

ROOT MATCHERS
isFocusable()
isTouchable()
isDialog()
withDecorView()
isPlatformPopup()

OBJECT MATCHER
allOf(Matchers)
anyOf(Matchers)
is(...)
not(...)
endsWith(String)
startsWith(String)
instanceOf(class)

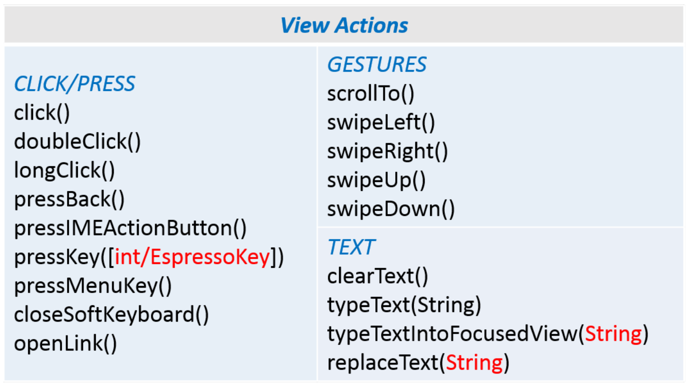
Figure 10.1: Espresso view matchers.

Espresso: It is considered as the entry point for interacting with views (through onView and onData). Also, presents APIs that are not connected to any view (e.g. pressBack).

ViewMatchers: A group of objects that implement ***Matcher< ? super View >*** interface. One or more of these objects can be passed to the ***onView*** method, it is used to locate a view within the current view hierarchy or to check if a view matches a certain criterion, for example, has a particular text.

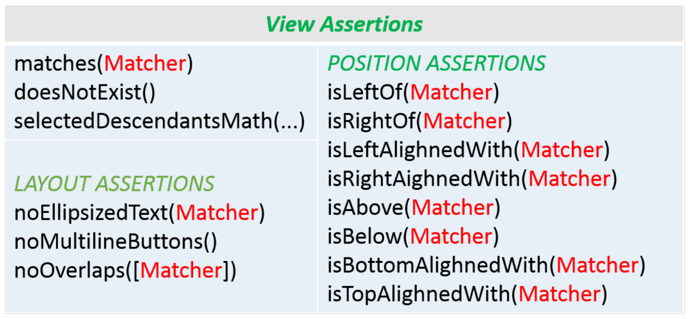
Espresso provides many ViewMatcher options (see Figure 10.1) that enable a mobile tester to effectively and uniquely locate views. These ViewMatchers give mobile tester ability to combine and create a combination of View Matchers to locate view uniquely. In Espresso it is easy to find the view very effectively even though with help of these ViewMatcher options.

ViewActions: A group of ViewActions that can be passed to the method ***ViewInteraction.perform()*** (for example, clicks, swiping, type text) and enable the tester to



View Actions	
CLICK/PRESS	*GESTURES*
click()	scrollTo()
doubleClick()	swipeLeft()
longClick()	swipeRight()
pressBack()	swipeUp()
pressIMEActionButton()	swipeDown()
pressKey([int/EspressoKey])	*TEXT*
pressMenuKey()	clearText()
closeSoftKeyboard()	typeText(String)
openLink()	typeTextIntoFocusedView(String)
	replaceText(String)

Figure 10.2: Espresso view actions.



View Assertions	
matches(Matcher)	*POSITION ASSERTIONS*
doesNotExist()	isLeftOf(Matcher)
selectedDescendantsMath(...)	isRightOf(Matcher)
	isLeftAlighnedWith(Matcher)
LAYOUT ASSERTIONS	isRightAighnedWith(Matcher)
noEllipsizedText(Matcher)	isAbove(Matcher)
noMultilineButtons()	isBelow(Matcher)
noOverlaps([Matcher])	isBottomAlighnedWith(Matcher)
	isTopAlighnedWith(Matcher)

Figure 10.3: Espresso view assertions.

implement actions on the views. Figure 10.2 shows the view actions methods.

ViewAssertions: A group of ViewAssertions that can be passed to the method ***ViewInteraction.check()*** (e.g. isDisplayed()). Usually, the mobile tester uses the matches assertion, which uses a view matcher to inspect the state of the currently selected view or sometimes to stop testing if a particular property is not fulfilled. Figure 10.3 shows the view assertions methods list.

For simplicity, a developer may implement these shortcuts to refer to them:

- ViewMatchers "***locate*** something"
- ViewActions "***perform*** something"
- ViewAssertions "***inspect*** something"

The standard syntax for test is to locate a view (***ViewMatchers***), apply something to that view (***ViewActions***), and then inspect some view properties (***ViewAssertions***).

Most of UI elements locating done with the help of resource IDs, but there is also capabilities to locate elements via strings which connected to the element. Espresso API attains fluently the goal of testing. The first thing: find the view that tester wants to perform on. Espresso has the concept of *ViewMatchers*. Once a mobile tester finds the view then he can act actions to it (typing, clicking, and swiping). Further, a tester can do some verification via *ViewAssertions*. What type of view mobile testers are interacting with (eg. edit text, button) are not specified explicitly, simply a view with a specific id which tester is looking for should be declared.

Formula:
Basic Espresso test: The Pocket Code app. It has a category_looks button, when you click on it, it will assert that a text is completely displayed on the screen.

```
onView(ViewMatcher)
             .perform(ViewAction)
             .check(ViewAssertion);
```

Example: The following code snippet shows how the test class might invoke this basic workflow:

```
Espresso.onView(withText(R.string.category_looks))//a ViewMatcher
              .perform(click())//a ViewAction
              .check(matches(isCompletelyDisplayed()));//a ViewAssertion
```

- Espresso comes with a *ViewMatcher* - withText. In this case, Espresso start to look for the view with the particular text (R.string.category_looks).
- When Espresso have found a suitable matcher for the target view, it is possible to use the *ViewAction.click()* to click on it.
- Assertions can be performed on the currently selected view with the *check ()* method. The most used assertion is the *matches ()* assertion, it uses a *ViewMatcher* to inspect the state of the currently selected view and check if a view fulfills an assertion.

In Espresso testing, a mobile tester should specify the class name of the startup activity in the application that is under test. When writing a snippet of code for testing it is possible to make use of string names or IDs of the UI elements to locate them, which enable the mobile tester to write regular black-box test cases. In the course of GUI testing, Espresso has the following key features.

Espresso supports emulator and real device: It is possible to execute tests written in Espresso framework both in emulator environments and on real devices.
Android version support: Espresso is created to support all versions of Android. It supports Froyo (API level 8) up to the newer versions.
Version control support: As Espresso only provides an API, there is no built-in version control support.
Finish activity: Espresso has the method that supports the "back"-button of Android mobile devices by using *ViewActions.pressBack()*, and through that, the activity that a tester is currently navigating can be terminated through this API.

Change device settings: In Espresso, it is not possible to change smartphone settings, but it is possible to configure the settings that can be changed through key codes, for instance, the volume settings.

Scrolling: Espresso API introduces *ViewActions* component, with this component, there are methods that can be called for scrolling. For example, *swipeUp()*, *swipeDown()* and *scrollTo()*. Since swiping is the only method a user has for scrolling in a view, Espresso has a perfect support for simulating this.

Support for device keys: By using *ViewActions* component, a mobile tester can call *pressKey ()* method that can be used to send Android keycodes to the view that is being connected with.

Delay support: In Espresso, delays are implemented automatically through the framework, and therefore it is not recommended to add additional delays, especially to compensate for slow emulators etc.

10.4 Continuous Integration

Continuous Integration (CI) is one of the most vital practices in modern software development. It is one of the agile techniques that aims to raise the quality of software and minimize the time taken to integrate changes by continuously doing integration and testing, it is unlike the most traditional systems of integrating and testing. Continuous integration has received a wide adoption, and the increase of commercial tools and open source projects is a clear proof of its success. When a developer has involved in a software development project using a traditional technique, he will face something so-called integration hell, where the time needed to integrate the changes exceeds the time needed to make the changes. On the contrary, CI is the system of integrating changes frequently and in minor steps. These steps are insignificant and, if an error exists in the project, it is easy that it can be corrected immediately [86].

Continuous integration is good practice that comes with the agile software development approach. It generates faster feedback to the developers and leveraging the test automation. It is widely used among the organizations with the complete development process, which are generally using a build server for implementation of the CI. The continuous integration introduces a challenging aspect for the mobile application development since many automated tests have to run on real devices. Hence, application deployment and installation on the device should be also automated [87].

The key idea behind continuous integration is to execute a predefined group of steps which are based on a specific trigger; the trigger could be a new pull request in the version control system, or trigger of time, for example, for automatically creating a nightly build and running all test cases on it, then creating a report that can be inspected via a web interface by all interested parties, and possibly reporting the main points of the test run to a developers' IRC channel of the project. In short, continuous integration refers to the process of integrating all development work at predefined times or events in order to be tested and built automatically. The idea is that this allows identifying the development errors early in the process [87].

The main tasks of the continuous integration server for mobile application development are:

1. Retrieve the source codes from the repository (GitHub).
2. Compile and examine the source codes, create new binaries.
3. Connect to the real device or emulator.
4. Run the unit tests.
5. Set up external dependencies (databases, services, etc.).
6. Run higher level tests (integration, service or UI tests).
7. Generate the application package.
8. Generate the report.

After all application tests are passed by the CI build server, application package along with the UI tests can be sent to the device cloud. Running the test on the device cloud is more expensive and usually limited by maximum device testing hours and device reservation time slots. Therefore, automated tests on real device cloud are not executed so often.

10.4.1 Continuous Integration Using Jenkins

Jenkins is a free and open source, extensible CI system that enables developers to build and test software projects or control the execution of external jobs [87]. Jenkins is used for continuous integration; it is an open source tool that allows controlling the execution of arbitrary automation steps. Jenkins has the ability to inform users about the success or failure of builds. It can be executed via the command line or can be run on a web application server. Jenkins is easy to install and has an intuitive and robust user interface. Automated testing is an important part of any continuous integration environment; it allows team members to receive feedback about the state of the development in a dashboard view (not only one's own) and ensures through documentation of results that the tests succeed and are regularly executed, together with all kinds of other mechanisms and collections of statistics, for example, for automatic code quality evaluations or test coverage metrics.

10.5 Version Control System

Version control is a system that tracks changes in a computer file or group of files over time so that you can recall particular versions later. If a graphic or web designer wants to save every version of an image or layout, a Version Control System (VCS) is a recommended tool to use. It gives him/her ability to revert files back to a previous state, revert the entire project back to a previous state, compare modifications over time, check who last modified something that might be causing a problem, who create an issue and when, and more. In addition, developers can easily recover lost files and get all these services for very little overhead [86][88].

A version control system is an inevitable tool for projects development where more than one developer is shared and one of the best practices even if developers coding is solo. Further, even though it is possible to use CI without VCS in place, it is not recommended for developers to avoid it.

10.5.1 Git

One of the most common VCSs examples is Git; Git is an open source and free, distributed version control system introduced in 2005 to manage every project (small or large) with high speed and efficiency. Git tracks changes in files and coordinates work on those files among several developers. It keeps and thinks about information in a different way as compared to the other systems. The main difference between Git and any other VCSs is how Git manages its information [88].

Basically, most other VCSs keep information as a list of file-based changes. Such systems (CVS, Subversion, Perforce, Bazaar, and so on) manage the stored information as a group of files and the changes made to each file over time. Git thinks of or keeps its data in a different manner. Actually, Git thinks of its data more like a group of snapshots of a diminutive file system. However, every time the developer saves the state of his project in Git (i.e. commits), the Git system takes a snapshot of what all his files look like at that moment and stores a link to that snapshot [86][88].

Git is considered as an efficient VCS, for instance, when developer files have not changed, Git does not store the file again, it just stores a reference to the previous identical file. Git considers its data more like a stream of snapshots [88].

We use Git with a basic branching model as version control system. We work with different branches to easily manage and control each phase of the software development and it is considered as a perfect system when one or more developers have to collaborate on the same feature. Git defines a master branch, development branch, release branches, and branches for each feature and bug fix. Each time when the user story (new feature) is “Finished” the developer merges a feature as a whole into the development branch. Then, the developers have to generate the release branch after all testing cases are done successfully. After all the features are finished the developers can merge into a master branch to use tags to identify the version number. The basis for continuous integration is the master branch since it should always contain a ready to release project.

10.6 GitHub

GitHub[3] is the largest web-based host for version control repository or Git repositories which was originally released in 2008. A great percentage of all Git repositories are hosted

[3]https://github.com/

on GitHub, and many open-source projects utilize it for Git hosting and other benefits. It offers access control and several cooperation features such as issue tracking, feature requests, code review, and task management for a massive number of developers and projects. It provides all of the distributed version control and source code management features of Git. GitHub is not a direct portion of the Git open source project, so there is a good opportunity for developers to connect with GitHub at some point while using Git professionally [88].

Projects which are hosted on GitHub can be accessed and managed using the standard command-line interface for Git. GitHub also enables registered and non-registered users to navigate public repositories on the website. On the same account, GitHub offers both features for private and free repositories which are usually used to host open-source projects.

10.7 Gradle

Gradle is an open source build management system based on Groovy; it was introduced in 2013 as the new preferred build system for Android apps. Gradle is completely free since it is built upon the principles of Apache Ant and Apache Maven. Gradle was designed to support multi-project and multi-artifact builds which can expand to be very large and supports gradual builds. By using a smart technique, Gradle can determine which parts of the build tree are up-to-date; therefore any task depends upon those parts will not be re-executed. Luckily, Gradle has features that support download and configuration processes of dependencies and other libraries. Developers can retrieve these dependencies by using Maven and Ivy repositories. This enables reusing concept for the artifacts of existing build systems [89].

When Google released Gradle and Android Studio, they had some objectives in mind, they focused on making it easier for code reusability, build-variants creation, and build process configuring and customizing. On top of that, they thought about easy IDE integration and making the build system independent from the IDE. In case the developer runs Gradle from the command line or on a continuous integration system the results will be the same as running a build from Android Studio. The developer's team of Gradle emphasizes that utilizing a declarative; DSL-form based on a dynamic language has large advantages over using the procedural, free-floating form of Ant, or any XML-based form used by other build systems [86][89].

Chapter 11

Localization to Bidirectional Languages for a Visual Programming Environment on Smartphones

11.1 Challenges of Localizing a Mobile App

The most important issues in localizing software to bidirectional languages can be solved during the design phase of the mobile app itself. Mobile apps should ideally be designed with features that will authorize them to be adapted into as many locales as possible without great engineering changes. Here are the main challenges encountered in localizing apps into bidirectional languages [59].

Character encoding:

The mobile apps should be able to display bidirectional languages characters as well as accept languages input from users. It is highly recommended to use Unicode scheme.

Right-to-left and vertical text:

Right-to-left text for languages like Arabic, Persian, Urdu, and Hebrew, and vertical text for Asian languages, need specific UI layout. The right-to-left script will also require mirroring of direction-sensitive graphics, while vertical script consumes more screen space. In addition, mixed languages' strings (for example, as a result of brand names or hyperlinks, which usually remain in English) or bidirectionality require extensive extra adaptation to support these languages.

Mobile phone screen size:

One of the most important challenges presented by mobile phones is the limited screen size. Due to this issue, mobile apps' text should be very minimal and developers, UI designers, and translators should take into consideration the short user interface strings.

Font style for mobile applications:

Localized product might have various UI fonts' types than the original products. Local users will use the best font type according to the font capabilities of the particular target language. Font styles for text are discouraged in layout design such as special fonts, italic, or bold, because they may affect the readability of complex characters in some languages.

Sometimes, specific fonts are too small and may require being increased in size for reading-clearness. On the other hand, some fonts are too big and may require being decreased. There can also be a critical issue with special characters, for example, umlauts, cedillas and other characters, not rendering correctly.

Text expansion:

The software designer should leave enough space to allow for text expansion. Translation into Arabic and other bidirectional languages causes approximately a 30 percent text expansion [6]. A layout flexibility of most dialog boxes and screens is necessary. One of the reasons for the text expansion is that when the source strings containing abbreviations which do not have parallel abbreviations in the target languages, for instance in English language M/F abbreviations stand for Male/Female and there are no parallel abbreviations in the Arabic language for Male/Female, the same for the abbreviations such as (GUI, DPI, GB).

Regional standards:

Mobile applications should support bidirectional languages locale standards as dates, times, currency, addresses, phone numbers and addresses formats and calendar settings, as well as sorting and indexing rules.

Search and replace:

There are special search rules that are specific to bidirectional languages (for example, Hamza, a character in Arabic orthography used to represent the sound of a glottal stop, transliterated in English as an apostrophe).

Icons, symbols, and images:

Although the use of graphics and icons instead of text can create a more international look and feel, it can also create some problems in localization. For instance, icons represent fingers such as an OK sign or V-sign may mean different things to different cultures. Indeed not all icons, symbols or images have a broad meaning. Special consideration must be given to the meaning of each icon/symbol/image and even color in the target culture and make sure they are culturally appropriate.

11.2 Considerations for Localizing Bidirectional Apps

App sales should not be limited to just one market; mobile app stores are global, and so the app should be too. The app store is available in over 150 several countries, and the first step to reach this global market is to internationalize the app. A user who tries to download the app could be located anywhere. Today, customizing the app to be localized is a must.

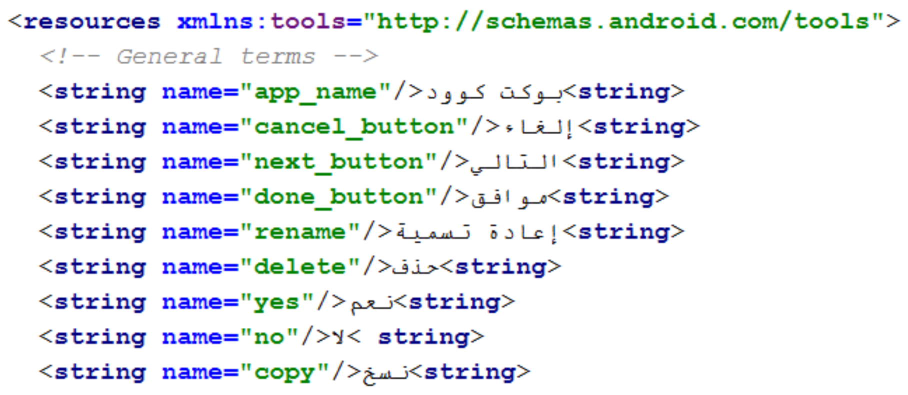
In practice, internationalization basically describes the process of preparing software app for localization. It is all about customizing app source code and design in a way to make all the text easily exchangeable. To do so, keep all text in separate resource files and reference them in app source code. On the other hand, localization requires complex and technical jobs accomplished by a variety of specialists, including engineers, translators, graphics designers, testers, programmers, and project managers. These considerations are originally targeted any Android based devices but could apply to any mobile phone operating system. The following are some best practices to consider for app BiDi localization.

11.2.1 Resources

Android will run on several smartphones in many locales. To reach the most users, the target application should handle text, layout, styles, numbers, currency, and graphics in ways appropriate to the regions where the target application will be used. A critical issue is that none of the UI element's text, colors, images, or styles should be hard-coded in the source code of the software (i.e., separating them from code). All of them should be stored in resource files; the content of resource files can be retrieved by the software at the run time to supply these elements to the users in their local language.

Externalize resources

All non-code assets related to an application are considered as resources, such as content, images, and videos. The localization process requires developers to add appropriate resources to a mobile app to ensure that a given country, locale, language, or culture is supported. Therefore, all resources are placed into external files. After that, localization can become a simpler process when creating new versions of the resources files for each supported language.



```
<resources xmlns:tools="http://schemas.android.com/tools">
  <!-- General terms -->
  <string name="app_name"/>بوكت كوود<string>
  <string name="cancel_button"/>إلغاء<string>
  <string name="next_button"/>التالي<string>
  <string name="done_button"/>موافق<string>
  <string name="rename"/>إعادة تسمية<string>
  <string name="delete"/>حذف<string>
  <string name="yes"/>نعم<string>
  <string name="no"/>لا< string>
  <string name="copy"/>نسخ<string>
```

Figure 11.1: String items for Arabic language.

Localizing strings. The user-facing text needs to be localized. Thus, the text needs to be translated to a specific language before it is displayed to the user. More specifically, all the strings should be stored in values resource files; the text can be retrieved by the software at the run time to supply these elements to the users in the Arabic language. Hence, an alternative XML strings file must be created - see Figure 11.1 . In Pocket Code/Paint's cases, the translation to Arabic is performed by volunteers using a community-based translation tool (Crowdin[1]). The string file that must be translated is uploaded to a web environment in order to be accessible to our translators. Practically, in Android, the UI elements are defined and stored in an XML file, and for every need-to-translate element, a string item is added in the XML strings file in the values directory [90]. The Arabic translation folder is created and named values-ar ("ar" represents the Arabic language according to ISO 639-1) [91].

Many tools for translating strings have been introduced; some are integrated with the development environment while others are platform-independent. Actually, translating strings automatically is not as succinct, complete, and correct as when a native speaker translates it. However, some words in the original language may have several meanings in the target language. For example, in some applications, there was a single-word text "kill," used to mean "stop the application." In Arabic, the literal translation of "kill" means "kill a person/kill an animal"; therefore skilled linguistic experts that also are domain experts for the app are needed for being able to correctly translate all strings. A translator must be qualified and have a base knowledge related to the project domain in order to translate this term as "stop" instead of "kill."

In real life, the miscommunication between developers and translators may produce many defects. In practice, the strings might be manipulated in the application without a translator's knowledge, and the translated string could then remain in an old state. Therefore, the developers must inform the translators about any changes in the string file and keep them up-to-date.

If developers need to distinguish between different regions that use, for example, the English language, they can add English regional variants for the United States, the United

[1]https://crowdin.com/

Kingdom, Australia, New Zealand, Canada, Hong Kong, and so forth to the app, and only the strings that are written in a different way would need to be included in the string file in order to be translated later; for example, “Colour” is favored in the UK, and “Color” is favored in the USA as shown in Figure 11.2.

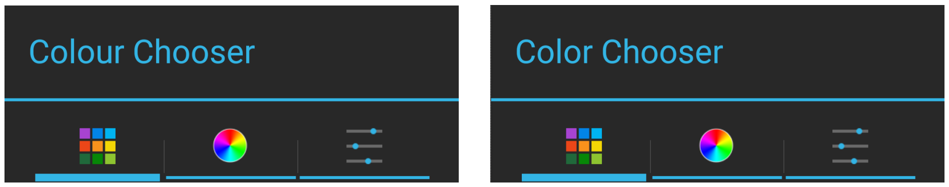


Figure 11.2: An example for regional variants: English (UK) versus English (USA).

Localizing layout. The layout is the directory which holding an XML file that defines the default layout. For example, if we need to provide some specific layout for the Arabic language and some generic layout for any other BiDi-languages (like Urdu or Persian), then alternative layout files should be created and placed within specially named subdirectories of the app. In Pocket Code, to be localized properly, some layouts need special adjustment (i.e. reversing UI elements with the view) as shown in Figure 11.3. When users run Pocket Code, the Android system selects which layout resources to load, based on the device’s locale.

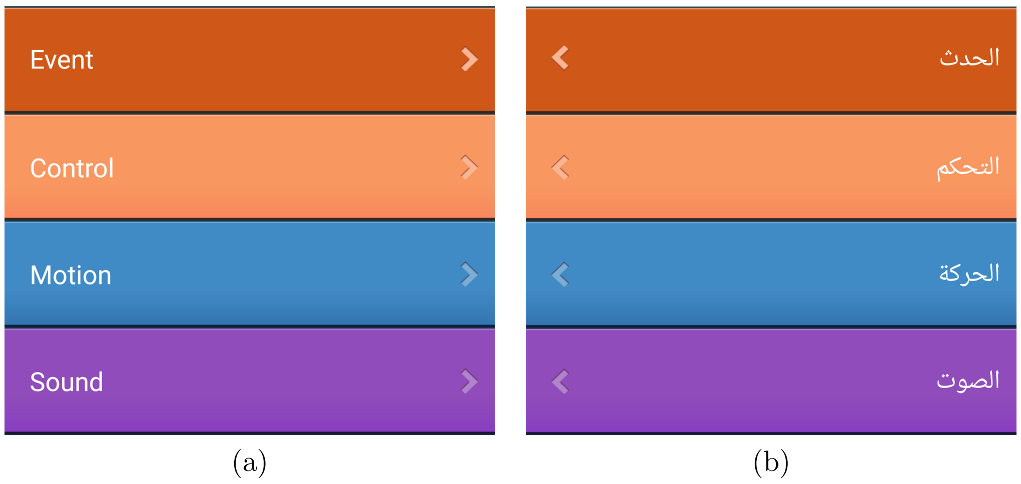
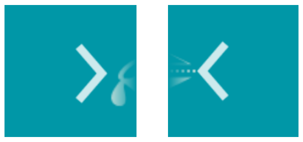


Figure 11.3: Screenshot for Pocket Code category’s menu(a) Original version (b) Localized Arabic version.

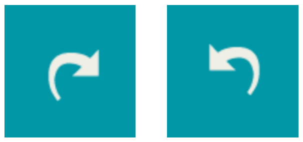
Localizing icons. Even though about 90 percent of the UI needs to be mirrored, not many of the icons, in fact, are affected. It is difficult to say which icons require mirroring and which do not because that depends on the icon’s semantics. Therefore, cooperation between designers, developers, and language specialists (e.g., translators) should be considered. Graphics which are direction-sensitive introduce another challenge with regard to mirroring. These graphics can get an incorrect meaning when mirrored.

Within an LTR layout in a navigator, an arrowhead that points to the left means that the navigation goes back to the previous page; an arrowhead that points to the right means that it goes forward to the next page. When the layout is mirrored for a BiDi, the meaning will be just the opposite because a mirroring just reorders the UI elements of a layout without mirroring the actual icons. Therefore, special attention should be given to direction-sensitive graphics such as icons that have a specific directional orientation (back, next, undo, and redo), help balloons, or toast messages. Figures 11.4 and 11.5 show the back, next, undo, and redo icons for an RTL language.



(a) (b)

Figure 11.4: Screenshots of (a) Back (b) Next for RTL languages.

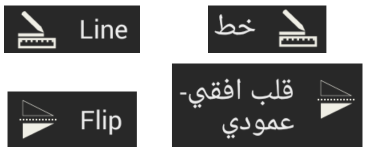


(a) (b)

Figure 11.5: Screenshots of (a) Undo and (b) Redo for RTL languages.

The solution for handling direction-sensitive graphics is to have different directories of graphics in your resources to be used when the drawing destination is an RTL direction, in other words, never hard-code images or layouts. Moreover, a complete optimization for RTL layouts is provided by adding entirely separate layout files using the ldrtl resource qualifier (ldrtl stands for layout-direction-right-to-left).

Further, for RTL languages the UI icons layout should naturally follow the RTL direction. A mirroring effect just replaces icons of a dialog box without mirroring the icons' images. This partial solution did not solve the layout problem in Pocket Paint's case for some icons such as the line and flip icons. It only gave them new positions within the mirrored dialog as shown in Figure 11.6.



(a) (b)

Figure 11.6: Screenshot of a partial mirroring. (a) English version. (b) Arabic version.

In this case the feature itself is mirrored to address the problem of UI layout as shown in Figure 11.7.

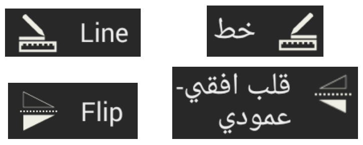


Figure 11.7: Screenshot of a complete mirroring. (a) English version. (b) Arabic version.

Localizing styles. A style is resource file that contains a set of attributes that define the look and format for UI elements. A style can specify view attributes such as width, height, margin, padding, text color, text size, gravity, background, and much more. All style properties are included in an XML resource file that is separate from the XML that defined the layout. To localize Pocket Code to the Arabic language, an alternative special styles resource file should be created.

The kind of style properties or values of default styles file should be customized to adapt and handle all RTL-languages peculiarities for different localization requirements such as padding, margin, layout direction, text direction, gravity. The following XML code shows samples of the default style attributes for the original language.

```
<item name="android:gravity" >left|center_vertical</item >
<item name="android:paddingLeft" >21dp</item >
<item name="android:paddingRight" >14dp</item >
<item name="android:textDirection" >ltr</item >
<item name="android:layoutDirection">ltr</item>
<item name="android:drawableRight" >@drawable/arrow_control</item >
<item name="android:layout_marginLeft">45dip</item >
<item name="android:layout_marginRight">45dip</item >
```

While the following code shows the Arabic version or any BiDi-languages versions.

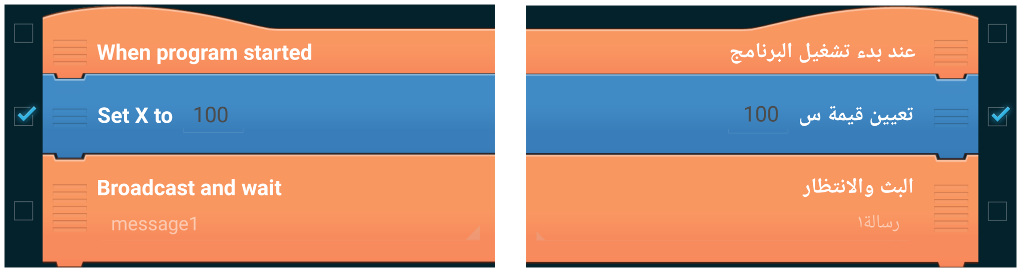
```
<item name="android:gravity" >right|center_vertical</item >
<item name="android:paddingStart" >21dp</item >
<item name="android:paddingEnd" >14dp</item >
<item name="android:textDirection" >rtl</item >
<item name="android:layoutDirection">rtl</item>
<item name="android:drawableLeft" >@drawable/arrow_control</item >
<item name="android:layout_marginStart">45dip</item >
<item name="android:layout_marginEnd">45dip</item >
```

At runtime, the Android system loads the appropriate set of styles resources according to the locale settings of the device. If the smartphone is set to Arabic language, Android will load properties from XML styles file, specifically, properties defined by styles Arabic version are applied to a single or multiple views in the layout, and all target views will be styled as defined by the styles file.

Default resources. The default resources of an app are those that are not marked with any language or locale qualifiers. If there is no default resource file, or if it is missing a string that the app needs to fall back on, then the app will stop and will show an error because Android looks for a resource and cannot find one that matches the configuration of the device.

11.2.2 Check Boxes

Check boxes and UI elements with check boxes are mirrored and right aligned, but the actual "check" symbols shall not be mirrored as shown in the Figure 11.8.



(a) (b)

Figure 11.8: Screenshot for bricks check boxes (a) Original version (b) Localized Arabic version.

11.2.3 Screen Size Variations and Limitations

One of the most important challenges presented by mobile phones is the limited screen size. However, if the app is designed to be used in different countries, both I18n and L10n test cases are especially essential under the constraint imposed by the different screen sizes. While designers may have designed their app to look fine in English, it may not look as well in German language or other languages where the character count will consume more space. On the other hand, some languages like Chinese contain a lot of information in each character and thus often need much less space, but this may not be true for other similar languages such as Japanese, since their characters include both Kanji that are of Chinese origin and purely phonetic Hiragana and Katakana characters, the latter two taking up comparatively more space. An additional problem with some of these Asian languages is that automatic line breaking can be very difficult as there are no space characters between words, and line breaks are not possible at all places, with additional complex rules to correctly break up lines.

11.2.4 Flexible Layout

Flexible layout is used to allow views relative to each other without fixed origins, widths, and heights. Any UI elements that contain text must be designed to be flexible and should resize properly. More space is allocated than necessary for English to accommodate most other languages (up to 50 percent more is normal). Practically, if fixed width constraints are used, localized text may appear truncated in some languages. Therefore, the constraints are removed and a "wrap content" attribute is used to allow limiting the width of each UI element.

Obviously, widgets and dialogs must be set to be expandable, both horizontally and vertically, in order to accommodate variations in texts width and height. The fonts in the Arabic language can expand horizontally and vertically more than in the English language. The result may be a truncated text because the space provided by the UI widget is not enough.

Arabic characters are more differentiated in structure and display than Latin characters. Sometimes translators may need two words or more in Arabic to describe one word in English. For example, in Pocket Paint, there is a single-word string "Zoom." In Arabic, the literal translation of "Zoom" consists of two words and clearly does not fit into the text view that was reserved for the English "Zoom," but there is no shorter word with the same meaning in Arabic. The same is true for "Ellipse" icon as shown, as also shown in Figure 11.9.

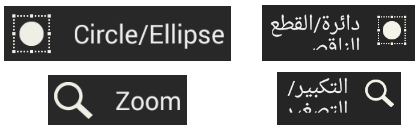


Figure 11.9: An example of a layout problem English >Arabic.

11.2.5 Font Style for Mobile Applications

Font styles for text are discouraged in layout design such as special fonts, italic, or bold, because they may affect the readability of complex characters in some languages.

11.2.6 System-Provided Formatting Methods for Dates, Times, Number, and Currencies

The system formats are used to specify dates, times, numbers, currencies, and other values that can be changed by the so-called "locale," thus ensuring the correct formatting of data according to the locale. If there is a specific format that is based on assumptions about the locale of users, the problem will arise when the user changes to another locale. Typical problems are "../../...." date formats between the US and European date formats, where it is unclear whether 10/2/2016 is the 2nd of October (US) or the 10th of February (most European countries) 2016.

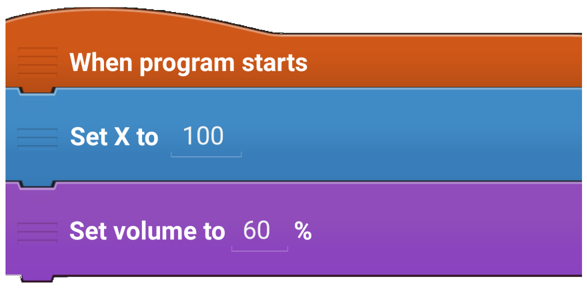


Figure 11.10: Screenshot for Bricks show "Set" word.

11.2.7 Importance of Textual Contents

Because of the small screen size of mobile phones, a lot of short phrases and words are suggested which lack content as well as words with multiple meaning, for example, "set." as shown in Figure 11.10. This can cause ambiguity and errors during the localization process. Therefore, to localize an app, the whole localization team should be very familiar with the application domain.

The string file for Pocket Code contains around (526) distinct string items (need-to-translate elements). During the translation process for the Arabic language, we face many troubles since some string items in the original language are ambiguous for translators and might be translated incorrectly and some strings in the original language may have several meanings in the target language. However, we make a small survey to seek for and locate how many string items that might be translated incorrectly or may have some type of ambiguity for translators. Practically, we find that there are around (100) string items that need more clarification such as (move, floor, stamp). Therefore, by providing Crowdin system with screenshots, we can eliminate the ambiguity or any misleading translation and the translators can know what the context of each string item is. In addition, sometimes the translator cannot distinguish if the target string item in verb or noun phrase to correctly translate it such as (measure, play, land), so by checking the screenshot of the app, translators can read the context and start translating in a proper way.

The quality of translations can be improved by providing screenshots of those terms which are subject to translation. Context conveyed by means of app screenshots eliminate the ambiguity of used strings and therefore translation errors, e.g., the term power could be interpreted as (electric) energy, physical strength or authority/prevalence in terms of influence or to rule over something. With a simple screenshot, e.g., showing a battery symbol it is immediately clear to the translator which interpretation is appropriate. Context deals with the semantics of terms being translated. By default as context information, the name of the resource string is shown in the Crowdin GUI which can help when it is wisely chosen but sometimes it is not enough as practice has shown. By the time of writing it is already possible in Crowdin to provide screenshots as context information in a limited manner as shown in Figure 11.11.

Uploading screenshots to Crowdin is highly recommended for localization projects especially. Although there is no API to automatically provide screenshots as context, it can be

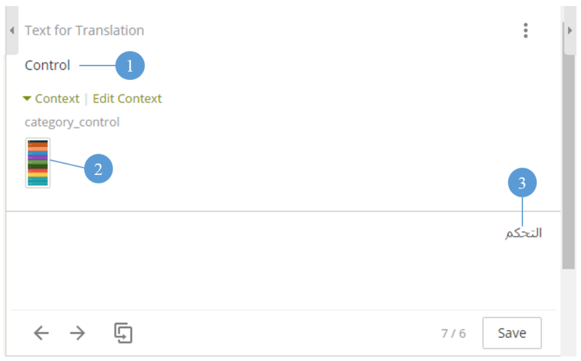


Figure 11.11: Screenshot for translation page in Crowdin system (1) string item (2) screenshot (3) translation in Arabic.

done manually by means of the Crowdin GUI. Therefore in our project (Catrobat), it is used only for ambiguous strings and where translators additionally requested clarification via the Crowdin GUI.

With the help of computer's mouse, localizer can highlight the text for translation. Crowdin will try to recognize either any strings uploaded to Crowdin match the selected string. In case, several similar strings were detected, there is an option to select the most appropriate one basing on context. Uploaded screenshots appear under each separate string in the Editor. Translators might have a look on them in order to know the exact context of the string.

11.2.8 User Interface Mirroring

The UI of a BiDi-language is generally mirrored and right aligned. Naturally, the common reading order for the speakers of BiDi-languages is from RTL. That is because those languages are written in a form known as RTL and strings flows in that direction as shown in Figure 11.12.

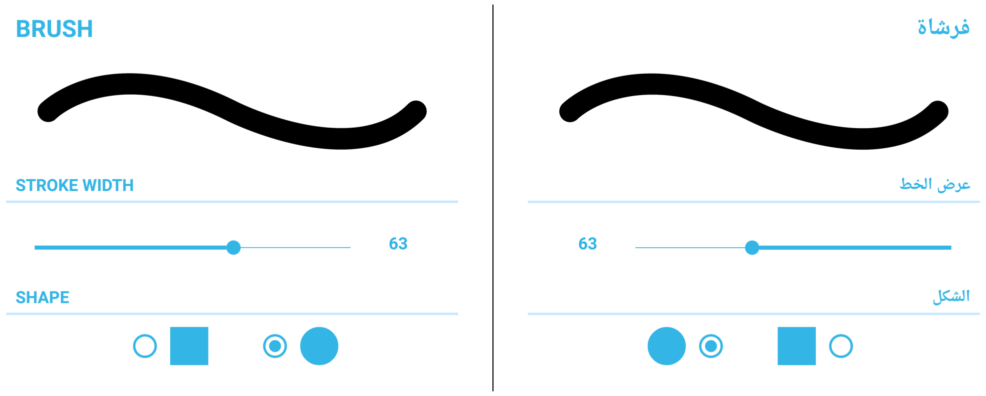


Figure 11.12: Example of brush's dialog in English and Arabic with bidirectional support.

In general, all UI elements should be mirrored, which includes lists, scrollbars, progress bars, pop-up boxes, grids, and galleries. The UI will be automatically mirrored when the user changes the system language to an RTL language. The direction of text is also changed to an RTL with the exception of phone numbers, country codes, and similar numbers which are by default LTR also in RTL languages. Note that the direction of some views and widgets in the UI layout may not change automatically.

In addition, content like images, video, and maps are not mirrored. Nevertheless, some directional images such as arrows need to be mirrored. The types of controls that should not be mirrored in BiDi-language are as follows:

1. Video controls and timeline icons since they represent the direction of the tape.
2. Images (except if they indicate direction or order).
3. Clocks.
4. Graphs (x- and y-axes in RTL are the same as in LTR).
5. Music notes.

11.3 Localizing Catrobat into Bidirectional Languages

Smartphone apps localization has still not reached its full potential. One of the reasons why there are many smart phone's apps in App Stores are not localized is due to the developing rate of this field. Every day more and more apps loaded to the market, and companies which want to keep up do not enough consider internationalization and localization activities during the product development phase.

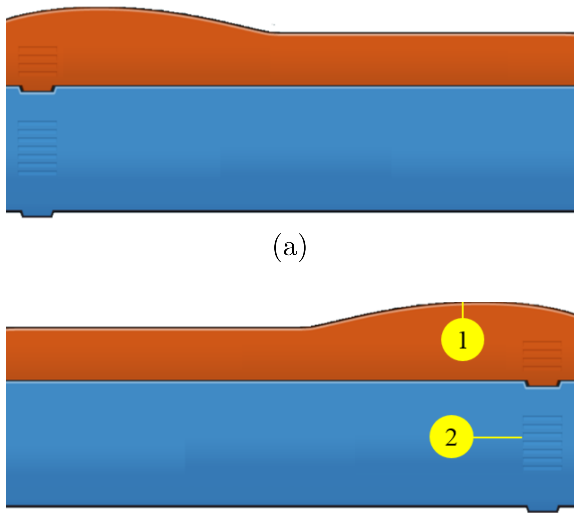
In this section, we provide all the challenging aspects of localizing Catrobat into bidirectional languages and achieve comprehensive solutions to the many and correlated challenges presented by such languages. Software localizers understand the challenges of localizing an LTR VPE into RTL. Therefore, we work carefully with source codes to comply with bidirectionality design guidelines and provide a consistent right-to-left look and feel product that suits the children needs and expectations. In particular, visually appealing, easy-to-use UI and graphics are our default objectives in localizing Catrobat into bidirectional languages.

11.3.1 Mirroring Bricks' Background

The direction of reading and writing influences how information should be drawn on the screen (i.e. mirroring awareness). Layout mirroring is a term used to describe the ability of an app to reorder the UI elements to match the right to left rendering for bidirectional locales. This requires, not only the text alignment and text reading order flow from right to left, but also the UI elements layout. The requirement for mirroring occurs since the text in such languages is read and written from right-to-left rather than left-to-right [92].

The background's graphic for each brick in Pocket Code communicates direction as shown in Figure 11.13 (a). Hence, to provide a consistent right-to-left look and feel to the brick layout features, the background should be automatically mirrored when its layout direction is right-to-left. In particular, the brick's layout is displayed flowing from right to left (see Figure 11.13(b)), and the following variations occur.

1. Curved area flows from RTL
2. Touch target is aligned to the right



(a)

(b)

Figure 11.13: Screenshot for bricks (a) LTR (b) RTL.

Enabling mirroring in resources

The background for each brick should reverse its horizontal orientation for right-to-left layouts; developers can include the mirrored images in a drawable-ldrtl resource directory. Particularly, in Pocket Code, the background for each brick is a 9-patch drawable. A 9-patch drawable enables the user to create bitmap images that automatically change the size to accommodate the views' contents and the size of the smartphone's screen [93]. The border is used to define the stretchable and static sections of the image. A stretchable section is defined by drawing one (or more) 1-pixel-wide black line(s) on the left and top part of the border (the other border pixels should be fully transparent or white) as shown in Figure 11.14. Therefore, due to these special attributes for the 9-patch drawable, the localization for drawable files (i.e. creating the custom version of drawable) in Pocket Code is an infeasible solution.

Enabling mirroring in the XML file

The (scaleX) property is used to scale of the view in the x direction. When X scale factor equals -1 (i.e. x values increase from right to left), UI element will be flipped horizontally.

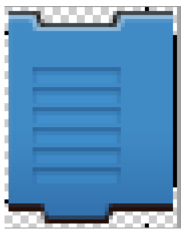


Figure 11.14: Example of 9-patch drawable in Pocket Code.

In XML file which defines the layout of Brick, the (scaleX) property is configured as shown in the below XML code:

```
<org.catrobat.catroid.ui.BrickLayout
      android:id="@+id/brick_broadcast_layout"
      style="@style/BrickContainer.Event.Medium"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:orientation="horizontal"
      app:horizontalSpacing="@dimen/brick_flow_layout_horizontal_spacing"
      app:verticalSpacing="@dimen/brick_flow_layout_vertical_spacing"
      android:scaleX="-1">
```

The main drawback of this solution is that, for each implemented brick layout, localizers need to create an alternative resource file and place it within specially named subdirectories of the Pocket Code. Manually repeating this for each implemented or new brick would be error prone and a waste of time. To gain more flexibility, the second way - mirroring in source code - is used instead.

Enabling mirroring in source code

There is a bug in 9-patch drawable with auto-mirroring, where it breaks the transformation matrix for drawing the child view afterward. The challenging aspect here is to mirror the background for each brick without mirroring the UI elements. Therefore, the Java class file which defines the layout of brick should be modified. The source code enables the localizers to get the background only and then perform the mirroring technique. Practically, we need to detect the reverted matrix, reset it, mirror the background of brick and then draw it again.

The following snippet of code illustrates how the matrix is detected and reset, and the dispatchDraw () method shows how the mirroring and drawing methods are implemented. By keeping in mind that this method is used when the Android version is prior to KITKAT.

```
@Override
   protected void onDraw(Canvas canvas) {
      float[] values = new float[10];
      canvas.getMatrix().getValues(values);
      if (values[0] < 0.0f)
         canvas.setMatrix(new Matrix());
      super.onDraw(canvas);
   }

   @Override
   protected void dispatchDraw(Canvas canvas) {
      if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.JELLY_BEAN_MR1 &&
            Build.VERSION.SDK_INT < Build.VERSION_CODES.KITKAT &&
            getLayoutDirection() == LAYOUT_DIRECTION_RTL) {
         Drawable background = getBackground();
         int color = getResources()
               .getColor(R.color.application_background_color);
         canvas.drawColor(color);
         canvas.save();
         canvas.translate(background.getBounds()
               .right - background.getBounds().left, 0);
         canvas.scale(-1.0f, 1.0f);
         background.draw(canvas);
         canvas.restore();
      }
      super.dispatchDraw(canvas);
   }

   protected void allocateLineData() {
      lines = new LinkedList<LineData>();
      for (int i = 0; i < linesToAllocate; i++) {
         allocateNewLine();
      }
   }
```

Android 4.4 (KITKAT) offers a new feature for drawable mirroring for RTL layout. In practice, the system can automatically mirror background images for all bricks by calling setAutoMirrored (). When the smartphone's language is set to Arabic, the backgrounds will be automatically mirrored – see Figure 11.13 (b). The below snippet of code illustrates how we can mirror the brick's background using the setAutoMirrored () method.

```
protected void onFinishInflate() {
      super.onFinishInflate();
      Drawable background = getBackground();
      if (background != null) {
         if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.KITKAT) {
            background.setAutoMirrored(true);
         }
      }
   }
```

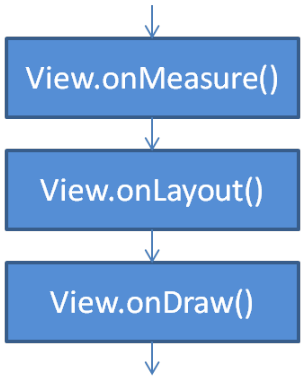


Figure 11.15: The three processes for drawing custom layout manager.

11.3.2 Reordering View's Children

The Android framework supports several default views but a developer can also create their own custom views and call them in their application. Views are typically created to provide high user-interface capabilities and experience which are not possible with the built-in views.

In Android, a view group is a special view that can hold other views. Sometimes, because of specific nature of requirements, the standard layout managers are not enough or developers are not satisfied with the existing functionality of any of the available layout managers. Therefore, developers need to extend the view group class to create their own custom layout manager to suit their needs [94].

Views are capable to measure, lay out and draw themselves and their child elements (in a case of a view group). Views are also providing the ability to save their UI state and handle touch events [94]. Parents are drawn before their children and children are drawn depending on the order when they are called. Drawing custom layout manager is done by passing three processes: measure, layout and draw as shown in the below Figure 11.15.

For all these features seen so far, in Pocket Code, the brick layout is implemented as a custom view since various design experience and functionalities are needed as shown in Figure 11.16(a).

User interface design is the most critical issue in bidirectional languages app. However, when designing for children and young people, developers often think that highly visual interface, simple interaction is the way to grab their interest [95]. For bidirectional languages, not only does the text alignment and text reading order render from right to left, but also the UI elements layout should follow this natural direction of rendering. Of course, this layout change would only apply to localized bidirectional languages.

When mirroring the brick's layout, padding and margin around UI elements and text also switch placement to match RTL layouts. However, in Figure 11.16 (b), the text view for "Place at" is aligned to the right, and the size and position requirements for the view and all of its children should match the LTR version (see Figure 11.16 (a)) as follow:

- Touch target height: (h) dp
- Screen edge margin before first UI element: (x) dp
- Text view bottom padding: (p) dp
- Text view left padding: (r) dp
- UI elements height: (m) dp

However, the onMeasure () method should be customized to introduce new measurements that support child laying out from right-to-left and adjust the brick's children alignment to the right. More specifically, the variable (posX) that determines where is to start laying out should be modified to handle both modes (LTR and RTL) based on mobile device's locale as shown in the below snippet of code.

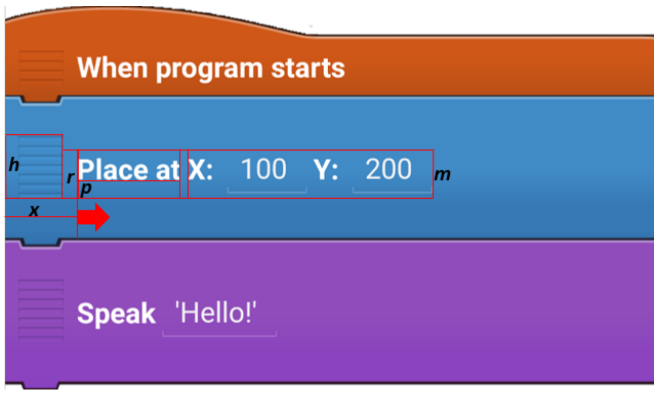
```
Configuration config = getResources().getConfiguration();
          if (config.getLayoutDirection() == View.LAYOUT_DIRECTION_RTL) {
             posX = sizeWidth - lineLengthWithHorizontalSpacing;
          } else {
             posX = getPaddingLeft() + lineLength - childWidth;
          }
```

In onLayout () we need to call layout method on each child of this view group and provide desired position (relatively to parent) for them using the sizes computed in the onMeasure () as shown in the below snippet of code. However, Figure 11.17 shows the Arabic localized version for Figure 7.2.

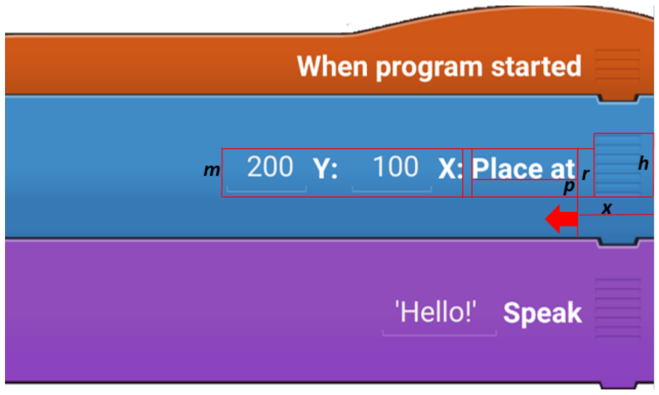
```
@Override
  protected void onLayout(boolean changed, int left,
                  int top, int right, int bottom) {
     final int count = getChildCount();
     for (int i = 0; i < count; i++) {
        View child = getChildAt(i);
        LayoutParams layoutParams = (LayoutParams) child.getLayoutParams();
        child.layout(layoutParams.positionX, layoutParams.positionY,
             layoutParams.positionX + child.getMeasuredWidth(),
             layoutParams.positionY + child.getMeasuredHeight());
     }
  }
```

11.3.3 Bidirectional Characters Rendering

Bidirectional text such as Arabic requires special rendering since there are different characteristics of scripts rendering include bi-direction, shaping as per context, reordering and linking characters [71][92]. In Pocket Code you can display text containing English, German, French, and Spanish string all at once. But, there are several scripts that need special processing to render and edit since the characters are not laid out in a simple



(a)



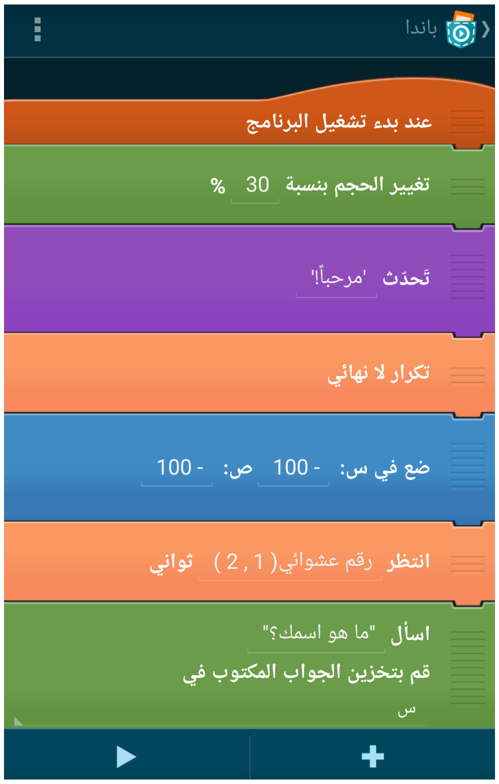
(b)

Figure 11.16: LTR vs. RTL laying out for bricks.

linear progression from left to right, as most European scripts are. These writing systems are referred to as "complex scripts."

To investigate more in this issue, a program which contains bricks for displaying text on the program's stage is created. In Figure 11.18, the variable (text1) is initialized to the English text, while the variable (text2) is initialized to Arabic. However, when the program is executed, the stage correctly displays the whole English text and oddly does not display any Arabic text as shown in Figure 11.19 (a). That is because Pocket Code, unfortunately, does not support bidirectional characters rendering on the stage and deliver its own font. Therefore, to make Pocket Code work properly for complex scripts when displaying typed text, the character should not be outputted one at a time, the text should be saved in a buffer and then displayed as a whole on the screen.

In Pocket Code, the bitmap font is used to draw characters on the stage. This technique does not support displaying of complex scripts. When language scripts are rendered from right to left, we might have to use other techniques than just drawing characters next to each other on the screen to produce something intelligible. In particular, to support complex bitmaps font rendering such as Arabic, Urdu, Persian, we need some way to convert the bitmap image that contains the text to texture object [96]. However, we use



باندا

عند بدء تشغيل البرنامج

تغيير الحجم بنسبة 30 %

تَحدّث 'مرحباً!'

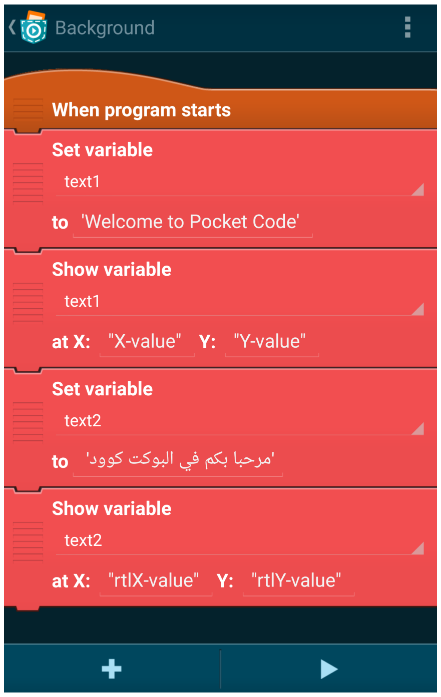
تكرار لا نهائي

ضع في س: 100 - ص: 100 -

انتظر رقم عشوائي(1 , 2) ثواني

اسأل "ما هو اسمك؟"
قم بتخزين الجواب المكتوب في
س

Figure 11.17: Screenshot for localized BiDi Pocket Code's program.



Background

When program starts

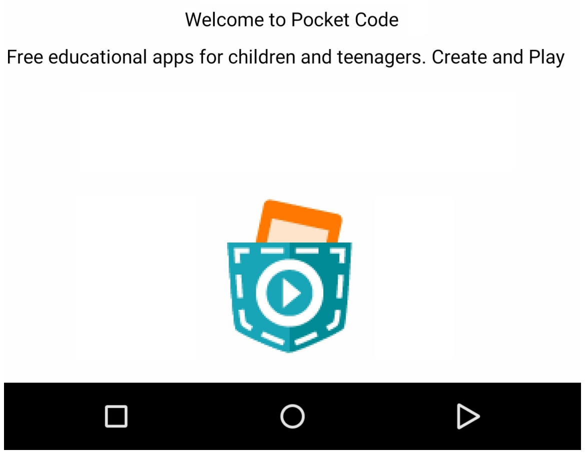
Set variable
text1
to 'Welcome to Pocket Code'

Show variable
text1
at X: "X-value" Y: "Y-value"

Set variable
text2
to 'مرحبا بكم في البوكت كوود'

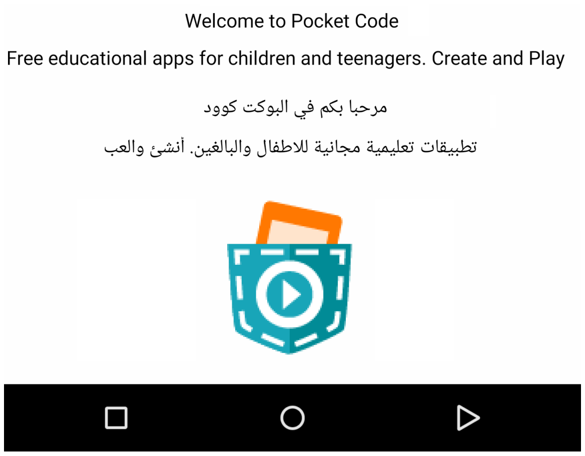
Show variable
text2
at X: "rtlX-value" Y: "rtlY-value"

Figure 11.18: Screenshot for the set and show variable bricks with English and Arabic strings.


Welcome to Pocket Code

Free educational apps for children and teenagers. Create and Play

(a)


Welcome to Pocket Code

Free educational apps for children and teenagers. Create and Play

مرحبا بكم في البوكت كوود

تطبيقات تعليمية مجانية للاطفال والبالغين. أنشئ والعب

(b)

Figure 11.19: Screenshot for Pocket Code's stage (a) Displays Latin scripts (b) Displays Latin and BiDi characters.

a single texture object to render all the glyphs. In order to draw the bidirectional text, we load the characters, upload them as a texture, and draw them at the correct offset from the starting position. When smartphone is set to the Arabic language, the Arabic version of Pocket Code correctly displays the text on the stage with proper morphology and directionality as shown in Figure 11.19 (b).

11.3.4 Bidirectional User Interface Mirroring

One of the most localization challenging issues is that some UI elements are being drawn outside the desired area or outside the UI canvas in localized version because its coordinates are expecting the origin to be in the top-left of the screen instead of the top-right. In Pocket Code, when the Arabic user tries to show the details of sprites, no details will be displayed on the screen. This defect is due to the localization process, all the views' positions in

Figure 11.20 (a) are changed to be drawn outside the screen and no details will be shown.

To support RTL mode, we need to provide some specific details view layout for any bidirectional languages, the custom version of the layout is created and placed within specially named subdirectories of the Pocket Code. Some code snippet is needed to adjust and correctly reorder the positions of text views by using the (*layout_toLeftOf*) instead of (*layout_toRightOf*).

Further, we use "start" and "end" instead of "left" and "right" in style file to provide some specific styles for the Arabic language. Also, for more precise control over the app UI in RTL mode, the attribute for setting the direction of a components' text is used: *textDirection="rtl"*.

In Figure 11.20 (b), title, icons, and UI elements are displayed flowing from right to left, however, when a UI layout is mirrored, these variations occur:

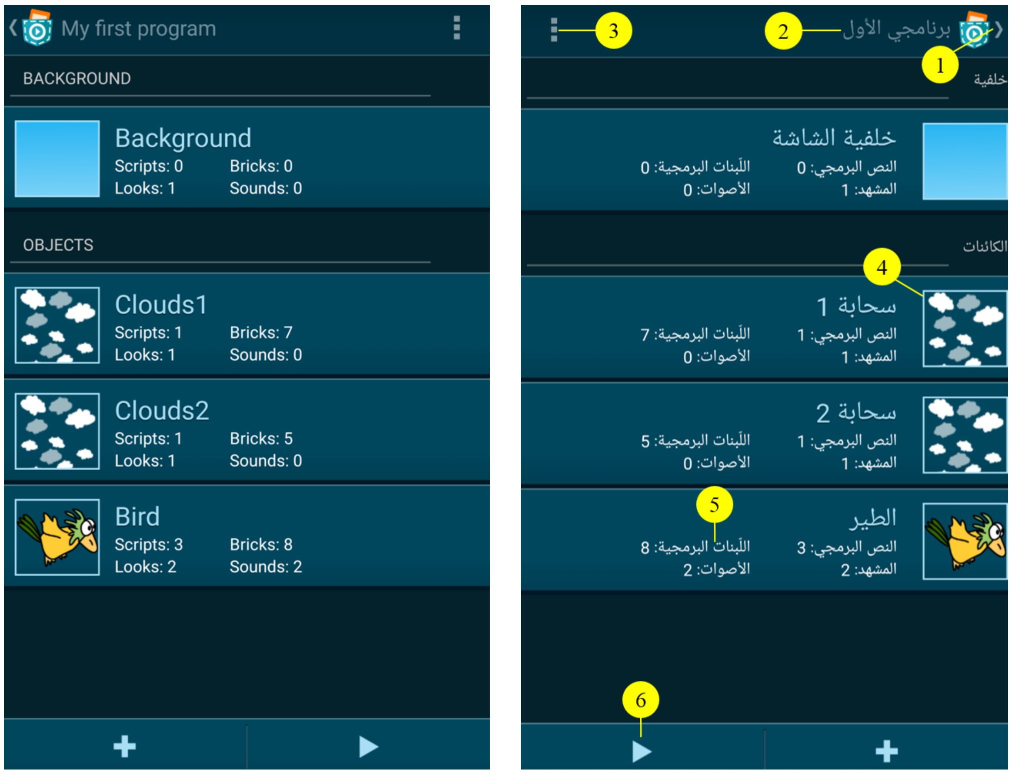
1. Back button points to the right.
2. Arabic text is right-aligned.
3. Menu button is left-aligned.
4. Icon appears to the right of text.
5. Scripts' field appears to the right of value.
6. Play button is not mirrored since it reflects the direction of the tape.

11.3.5 Generating Strings During Application Runtime

In the course of localizing software, all strings should be declared as resources and separated from the source code [6]. Typically, program size is expressed in units of measurement (KB, MB, GB, etc.). On projects' details screen (see Figure 11.21 (a)) these units are implemented as a hard-coded string since the size of each program should be computed, and then mapped to the proper unit of measurement at the program's run time. On an RTL screen (see Figure 11.21 (b)), the placement of unit ("MB") appears to the left of value, further, the following localization issues should be considered:

1. Size's value is generated at the run time.
2. "MB" unit is a hard-coding string (i.e. untranslated).
3. Hindi digits are used for numbering style.

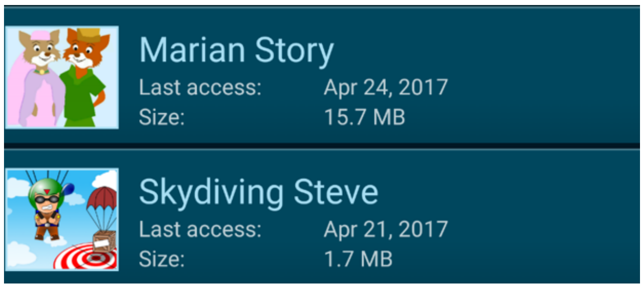
The developer has to be careful with strings that are composed at the application's run-time. To solve this issue, all of the strings should be moved into resource file; the content of the file can be retrieved by the software at the run time to supply these elements to the users in their local language. The following snippet of code is used to remove the hard-coding string of measurement's units.



(a) (b)

Figure 11.20: Screenshot for sprite-details screen (a) Original version (b) RTL screen.

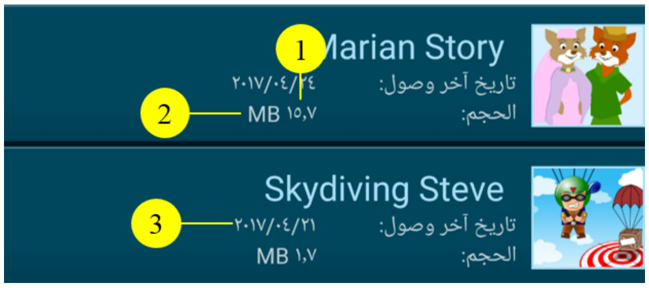
```
public static String formatFileSize(long bytes, Context context) {
     final double unit = 1024;
     String fileSizeExtension[] = new String[]{
          context.getString(R.string.Byte_short),
          context.getString(R.string.kiloByte_short),
          context.getString(R.string.GigaByte_short),
          context.getString(R.string.TeraByte_short),
          context.getString(R.string.PetaByte_short),
          context.getString(R.string.ExaByte_short)
     };
     if (bytes < unit) {
        return bytes + " " + fileSizeExtension[0];
     }
     int exponent = (int) (Math.log(bytes) / Math.log(unit));
     exponent = Math.min(exponent, fileSizeExtension.length - 1);
     String prefix = fileSizeExtension[exponent];
     return String.format(Locale.getDefault(), "%.1f %s", bytes /
          Math.pow(unit, exponent), prefix);
  }
```



Marian Story
Last access: Apr 24, 2017
Size: 15.7 MB

Skydiving Steve
Last access: Apr 21, 2017
Size: 1.7 MB

(a)



Marian Story
تاريخ آخر وصول: ٢٠١٧/٠٤/٢٤
الحجم: MB ١٥,٧

Skydiving Steve
تاريخ آخر وصول: ٢٠١٧/٠٤/٢١
الحجم: MB ١,٧

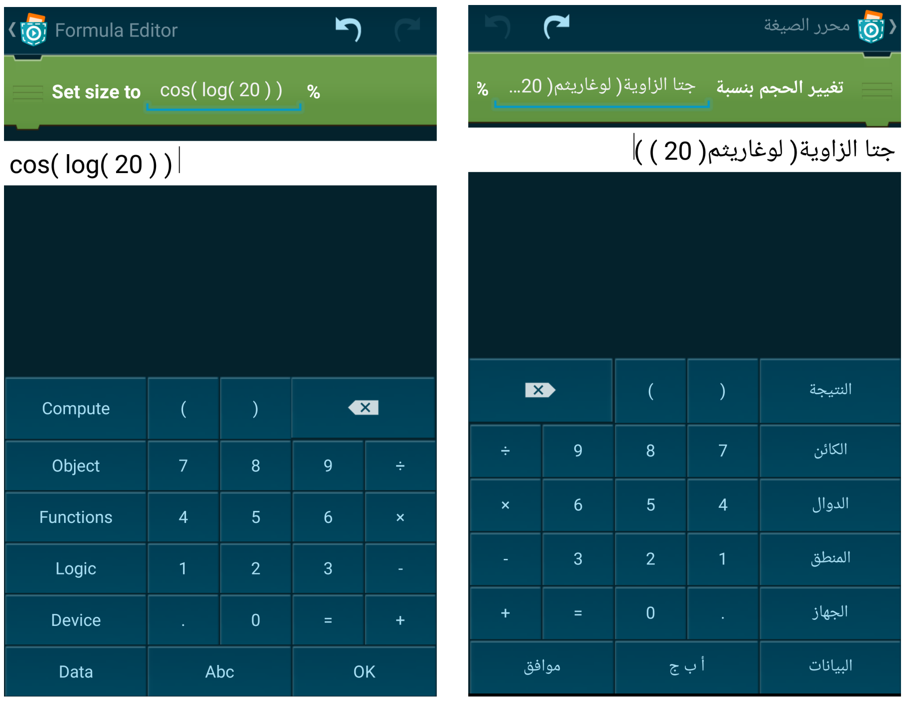
(b)

Figure 11.21: Screenshot for program-details screen (a) Original version (b) RTL screen.

11.3.6 Formula Editor

In Pocket Code, the formula editor was developed to calculate variables. It has an interface like a pocket calculator and enables users to use all available functions, logic, devices, and data as shown in Figure 11.22 (a).

The formula editor grid in Arabic and other bidirectional languages should be mirrored, and the highlight should be mirrored and right-aligned. As all progression in a bidirectional language should move from right-to-left and begin to the very right of the grid as shown in Figure 11.22 (b). Therefore, the alternative formula editor keyboard layout file is created and placed within specially named subdirectories of the Pocket Code. The default position in any edit text field in a bidirectional language UI is to the very right even before any character or text has been typed into the field - see Figure 11.22 (b).



a.) LTR b.) RTL

Figure 11.22: Screenshot for formula editor grid.

Chapter 12

Improving Pocket Paint Usability Via Material Design Compliance and Internationalization and Localization Support on Application Level

This chapter discusses the implementation of Google's Material Design guidelines, internationalization, and localization for mobile applications in the case of Pocket Paint, an Android painting application. The intended goal of this redesign is to broaden the user base by improving overall usability and supporting right-to-left written languages such as Arabic. The main challenges of the redesign are the intricacies to thoroughly support both right-to-left and left-to-right scripts, e.g., the positioning, translation, mirroring of text and graphical elements, the 'when' and 'when not' to mirror.

Related to the Material Design guideline compliance we carried out a user experience test with six users (age 13) of our target group. All participants rated the redesigned application being simpler, more appealing and concise in comparison to the previous version.

12.1 Introduction

Pocket Paint is a mobile paint editor developed by the free and open source non-profit Catrobat project, creator of the free educational Pocket Code app - a mobile app to visually create programs directly on one's mobile device. Pocket Paint supports transparency and zooming up to pixel level, which are not widespread functionalities for mobile paint apps. It is integrated into Pocket Code but can also be used on its own. Although there are no commercial interests behind Pocket Paint (it is open source software, free of charge and advertisement), a facelift was necessary since the app was designed complying with the

Android Design guidelines for the year 2012 and first released in 2013.

The Pocket Paint redesign was started for two main reasons. Firstly, we wanted to broaden the user base by multi-language and bi-directionality support. According to data collected by Google and Admob in March 2014, the number of users who have stopped using an app, because it was not localized properly, varies between 34 percent and 48 percent depending on the origin of the data (United States; China; Japan; United Kingdom and South Korea)[61].

We wanted to implement multi-language and bi-directionality support on application level since previous versions' support was only on operating system (OS) level. This means that an app's language and locale is changed according to system language settings. From a usability point of view, it is better to be able to set an app's language within the app itself, either default to the language of the OS, or independently. This is particularly true for poorly translated apps. Furthermore, there are languages, e.g., Sindhi or Pashto, which are not (yet) supported by the OS but which can be supported on an application level. Secondly, we wanted to comply with Material Design guidelines to improve the graphical user interface(GUI).Overall, both directions of redesign should contribute to an improved user experience in the sense of improving the look, feel and usability [97] of Pocket Paint.

12.2 Background

12.2.1 Material Design

Material Design2 guidelines are compiled and updated frequently by Google since 2014. They describe visual, interactive, and motion standards for mobile app and web development. Material is a metaphor, a system for uniting style, branding, interaction, and motion under a consistent set of principles [98]. The aim of Google Material Design is to create a unique visual design that incorporates all the company's services [99]. With its rich set of guidelines, principles and resources [100], it became the leading industry standard in development.

12.2.2 Internationalization, Localization, Bidirectional Script Support

Although internationalization and localization are colloquially often used synonymously, there is a subtle difference. I18n refers to the process of (re)engineering an application to support various languages without further modification, i.e., making the app world-ready. L10n refers to the process of adapting internationalized software for a specific region or language, i.e., making the app language and culture specific [6]. I18n and L10n require more than just translating user interface strings [17]. It should be considered from the very beginning of the application design since late changes are time-consuming and hence expensive. The following generic I18n items should be considered by developers [6].

- Locale and culture awareness (e.g., dates, times, addresses, units, and phone numbers)

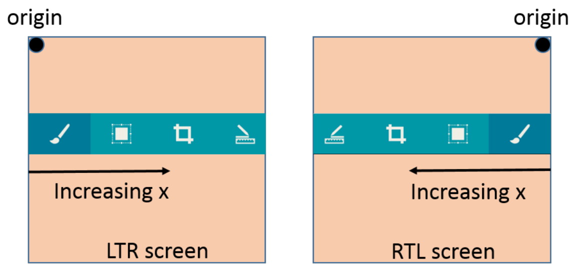


Figure 12.1: Layout mirroring for LTR and RTL screens.

- Numbering style (e.g., Arabic or Hindi digits)
- Layout direction (e.g., left to right in Western languages, right to left in Arabic, Hebrew and Persian)
- Character encoding scheme for textual display
- Case conversion
- Complex scripts awareness
- Sorting and string comparison

Localization deals with adaptation of software and content.

- Language translation text
- Layout direction (i.e., mirroring awareness)
- Direction-sensitive graphics (e.g., undo, redo)
- Spelling variants for different countries where the same language is spoken, e.g., localization en-US, en-CA vs. en-GB, en-AU
- Images and colors (i.e., cultural appropriateness)
- Names and titles

Localization adjustments should be done by people aware of culture and language for the specific region as well as being familiar with usability and software ergonomics.

12.2.3 How Does Bidirectionality Affect UI Design?

In bi-directional language software, text can flow in both directions. Western languages are written from LTR. Arabic and Hebrew, for instance, are written from RTL but numbers are written and read from LTR. The direction of reading and writing affects how information should be presented on the screen (i.e. mirroring awareness) [6]. In general, an RTL layout is the mirrored image of an LTR layout. Mirroring is done by coordinate transformation from LTR, where the origin is in the upper left corner, to RTL layout, where the origin is in the upper right corner of the screen (see Figure 12.1). Mirroring also affects the direction of some icons and images, especially those conveying a sequence of events [98]. For instance, the flow of time is depicted from left to right in LTR languages but vice versa in RTL languages. In RTL languages specific terms like hyperlinks, company or brand names are not mirrored or translated but written in the orientation of their origin. When a UI layout is mirrored for bi-directional languages, these variations occur [98]:

- Icons are displayed on opposite side of an input field
- Navigation buttons are displayed in reverse order
- Icons with a particular directional orientation, e.g., undo and redo
- Translated text is aligned to the right

The following types of items are not mirrored in bi-directional languages [98]:

1. Icons without a particular directional orientation, e.g., camera and floppy disk
2. Video controls and timelines (tape direction)
3. Numbers and phone numbers
4. Charts and graphs (x/y-axes same for RTL and LTR)
5. Musical notes
6. Images (except if they depict direction or order)

12.2.4 Multi-Language UI Support on App Level

For 2021 the shipments of smartphones with Android or iOS are forecast to reach around 1.8 billion units worldwide [101]. Due to this global situation, companies vending mobile applications can not afford to publish apps containing critical bugs or usability issues. If users have a bad experience, they will uninstall the app and switch to a competing product. If applications do not support different languages, they will not be used in countries where the apps' languages are not understood. This limits the potential user base significantly.

For instance, Arabic is spoken in more than 23 countries and more than 30 languages use scripts written from right to left therefore not only multi-language support but also bi-directional script support is needed. Culture-specific strings that are translated into the languages of the target locales are kept isolated from the rest of the app which facilitates the support for more locales and languages. For every language and locale, unique directories are created inside the app's resources folder. Alternative translation string files are stored in these locale-specific resource directories, containing graphics, sounds, layouts, and other locale-specific resources. The OS will load the appropriate resources according to the user language settings of the app at runtime.

When the language of an app is changed, the whole app is recreated. So if the app has any data object, the object's instance state must be handled (saved and restored). Figure 12.2 illustrates the sequence of steps that must be implemented to change the locale (language and country) according to the language preferences. The multi-language feature provides an interface to change the app's language independent of the device's language settings.

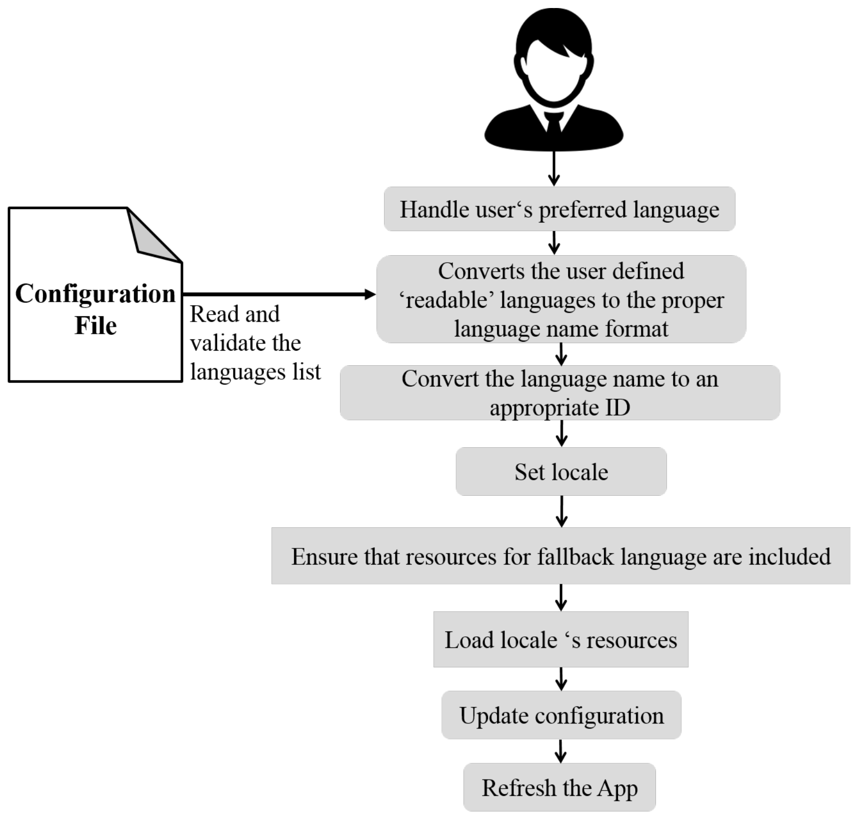

Configuration File
Read and validate the languages list
Handle user's preferred language
Converts the user defined 'readable' languages to the proper language name format
Convert the language name to an appropriate ID
Set locale
Ensure that resources for fallback language are included
Load locale 's resources
Update configuration
Refresh the App

Figure 12.2: The sequence of steps to change the locale (language and country) of an application.

Multi-language support on application level can support languages not (yet) supported by the operating system.

12.3 Redesigning Pocket Paint

12.3.1 New Features and Enhancements

The new version of Pocket Paint and its new features such as navigation drawer, landscape device orientation, layer support and feature discovery (help) now comply with the Material Design guidelines. The navigation drawer is used as a new main menu. The app is now aware of the device's orientation and changes the GUI between portrait and landscape accordingly. Layers have been introduced to enable the user to metaphorically create composite images from stacks of transparent slides. Feature discovery is an interactive help system shown at the first start of the app. This quick tutorial interactively explains the user the most important features of the Pocket Paint. Feature discovery can be skipped at any point and will not show up until triggered by the help item from the navigation drawer. Drawing tools have been enhanced to support more shapes and the color theme has been updated to a more brighter theme. The most important enhancements in the new version of Pocket Paint are multi-language and bi-directionality support on application level.

12.3.2 Bidirectional Design Guidelines for Pocket Paint

Localization is not only about different languages but includes text, layout, and graphics.

Text and layout

For bi-directional languages, the default direction is from right to left. The layout of all user interface elements should follow this direction. Therefore supporting bi-directionality needs layout customizations for text and also for all UI elements, including buttons, text views, seek bars, sliders, check boxes, menus, and dialog boxes.

Cultural attributes

All graphical controls must not include any specific cultural attributes or idioms. Graphical user interface components of a product need to be revised (made culture neutral) for the international market. Such attributes might be easily misunderstood by users of different regions, e.g., finger symbol icons for ‘OK’, ‘excellent’, or ‘victory’. During the localization phase, potential ambiguities can be easily adjusted. Also, the use of puns in user interfaces should be avoided since they are prone to be misunderstood by non native speakers. This will minimize the expensive localization needs for graphical elements. Icons that have a specific directional orientation such as back, next, undo, and redo should be given a special attention for bi-directional languages in order to be locale-aware. The localization team can present feedback about the graphic elements used in the product to ensure that they are convenient for all locales.

Icons related to flow of time

When text and icons are mirrored to follow the RTL layout, anything related to the flow of time should be depicted as moving from right to left, e.g., back and forward buttons. In RTL layouts, backward points to the right $\rightarrow$, and forward points to the left $\leftarrow$.

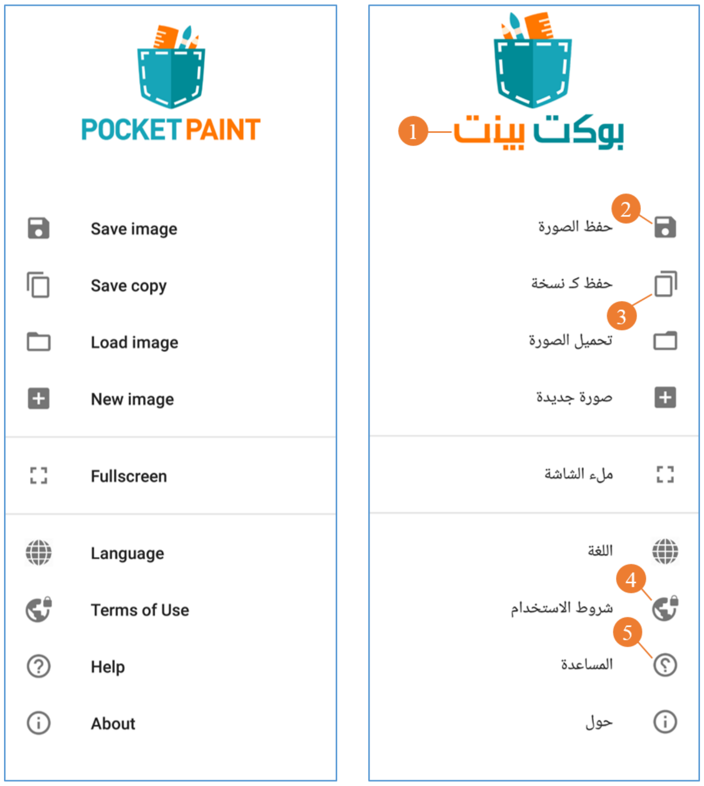
Scanning direction

Since the RTL languages are written from right to left, printed documents or screen of applications are read from the upper right corner. RTL readers start scanning the screen in the upper right corner (i.e., a right-to-left, top-to-bottom order) therefore the important information should be placed there.

Layout comparisons

In Figure 12.3 the comparison between left-to-right and right-to-left rendering of Pocket Paint's navigation drawer is depicted. On RTL screens, text, icons, and UI elements are displayed flowing from right to left, with following five intricacies:

1. The Arabic Localized logo is used.
2. Save icon appears right of the text - it should not be mirrored since it represents a real object (floppy disc).
3. Icons conveying direction are mirrored.
4. Icons not conveying direction are not changed.
5. The question mark is just reversed.



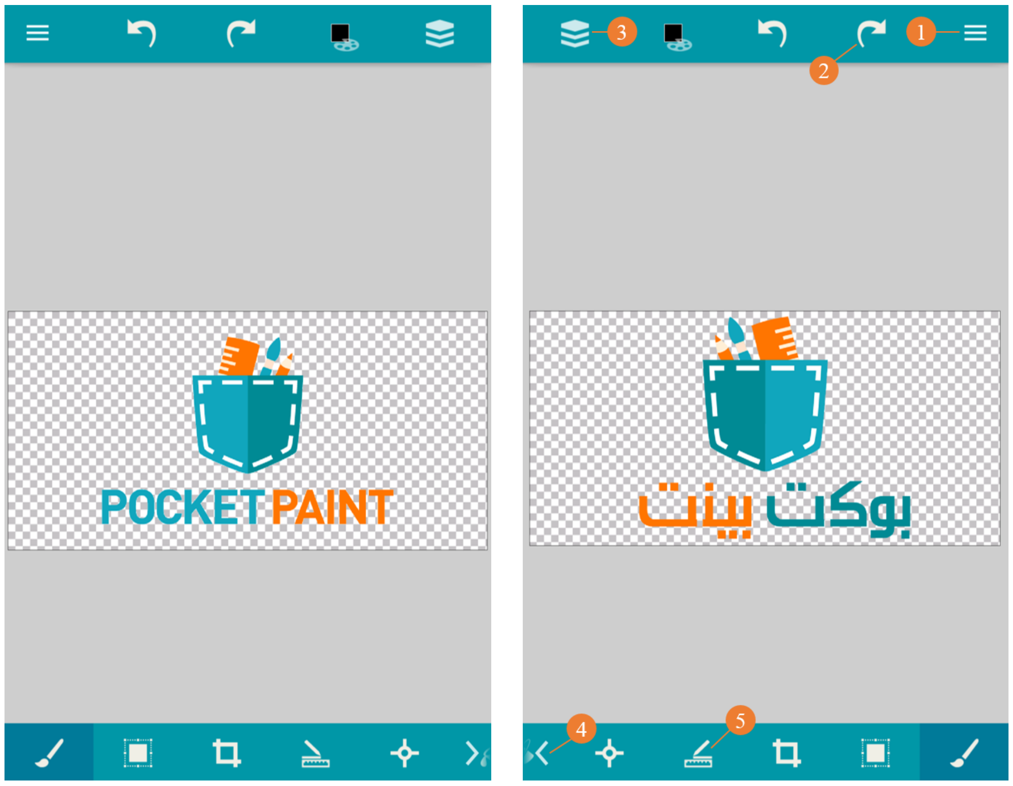
a.) LTR b.) RTL

Figure 12.3: Pocket Paint's navigation drawer.

For localization into RTL languages, the UI icons should follow the RTL direction by default. Sometimes, the circular direction of time is depicted in icons, e.g., undo and redo buttons. In Figure 12.4 a direct comparison between LTR and RTL layout of the drawing canvas is depicted. The intricacies are numbered from 1 to 5.

1. Navigation drawer button is right-aligned.

2. Undo icon is mirrored to follow the RTL direction.
3. Layer button is left-aligned.
4. Next button points to the left.
5. Icon that communicate direction is mirrored.

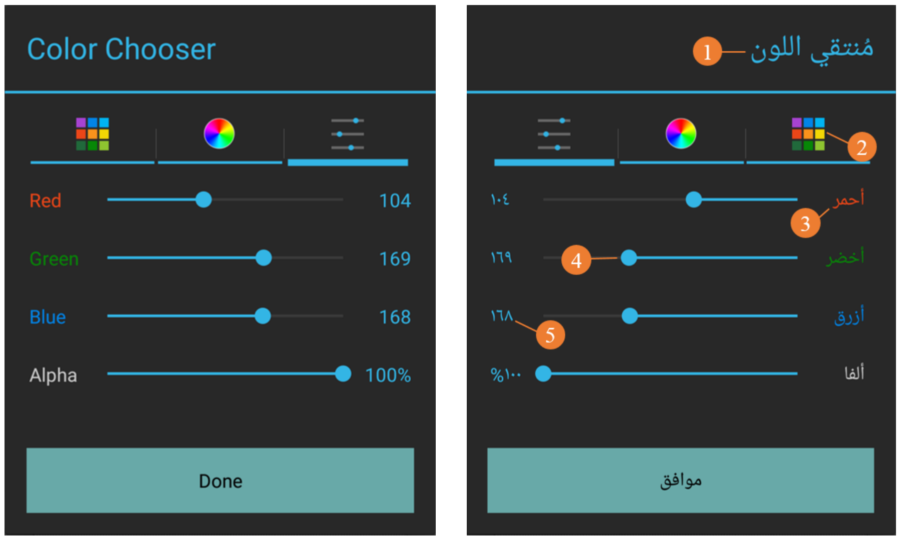


a.) LTR b.) RTL

Figure 12.4: Pocket Paint's drawing canvas.

The differences of the color picker LTR and RTL layout are shown in Figure 12.5. Focus the following intricacies:

1. Dialog's title is aligned to the right.
2. Color palettes is right-aligned.
3. Text appears to the right of slider.
4. Slider is mirrored and should progress RTL.
5. Hindi digits are used for numbering style.



a.) LTR b.) RTL

Figure 12.5: Pocket Paint's color picker.

12.4 User Experience Test

The goal of the redesign was to broaden the user base and to improve overall user experience (UX). The chosen means was to implement Material Design and multi-language and bidirectionality support on the application level.

As a first step, we carried out a UX test with six seventh grade students (age 13). The participants are within Pocket Paint's target user group, namely 13-17 years old teenagers. We chose the number of participants according to [102] which states three to six participants are sufficient to cover 80 percent of the usability problems. The students were using Pocket Paint in their art classes. All of them knew the previous version of Pocket Paint and therefore noticed the changes hence they could give feedback in relation to the new version. The UX test was designed to complete six small drawing tasks by creating certain picture elements. Every task was designed to use different drawing tools or aspects of the application. Completing these tasks resulted in similar images. The students were observed (recorded) during their work on the tasks and then asked for feedback. Qualitative judgment was done in school-grade like manner: A=1 ... best grade, B=2, C=3, E=4, F=5 ... lowest grade. In average the simplicity of the app was perceived as having improved from 3.2 to 1.8, and the optical appeal from 3.5 to 1.3. This is a hint that the redesign is a step in the right direction.

The students stated that the new version is clearly better and offers all features the old version had. Nevertheless during the test a couple of improvement points were isolated: a.) tools or functionality which could not be found within a short time so students needed guidance, for e.g., shapes, opacity, color palette, zoom as well as gallery and save, and b.) not or misunderstood functionality, e.g., image load vs. import, undo/redo functionality, deselection of tools, crop functionality. After analysis of the students' feedback the following action items were identified: a.) give hints how to use touch gestures, e.g., zoom drag, b.) improve visual feedback for placement of shapes and import graphics c.) rework of undo/redo d.) unify the appearance of the color dialog and finally e.) include the layer feature as well as the navigation drawer into the app's introduction and help. These items along with some minor bugs which were discovered during the test will be implemented and fixed until the official release in Google's Playstore in summer 2017.

Chapter 13

Bidirectional Languages Localization Testing Framework: Implementation and Analysis

After the product localization, localization testing is performed. The objective is to ensure that the localized product is fully functional, cosmetically correct, linguistically accurate, and culturally appropriate and that no issues have been produced during the localization process. Localization testing focuses on two general areas. The first involves things that are often changed during localization, such as the UI and content files. The second consists of language-specific, culture-specific, and country-specific areas.

Figure 13.1 illustrates the architecture of the proposed automated bidirectional localization testing approach. The proposed approach utilize best-practice GUI testing tools for Android (Robotium and Espresso), create new custom matchers and assertions to introduce a framework that able to support bidirectional localization issues, and invokes them via a continuous integration server in order to execute all automatic GUI tests from a central place. This chapter provides substantial, complete development and user satisfaction testing methods for the bidirectional localization aspects of the app.

13.1 View Matchers and Assertions for Bidirectional Localization Testing

Espresso is an extensible testing framework, it introduces flexible APIs for views and adapter matching, so the mobile tester can create his own matchers and assertions in target Android app. In the course of localizing a mobile app, a tester can extend the framework capabilities by writing an advanced set of view matchers and assertions to improve the current matchers and assertions available for mobile application testing.

Espresso enables mobile testers to create their own custom matchers and assertions, such matchers and assertions are really useful for automation testing. In the proposed frame-

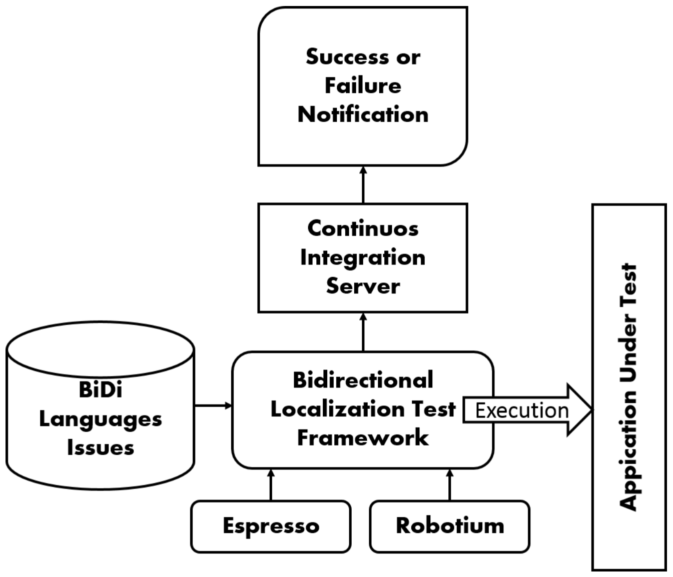

Success or Failure Notification
Continuos Integration Server
BiDi Languages Issues
Bidirectional Localization Test Framework
Execution
Appication Under Test
Espresso
Robotium

Figure 13.1: Localization testing architecture.

work, Espresso is customized in order to give the framework ability to efficiently support bidirectional languages issues. However, the new custom matchers and assertions are created to give tester flexibility in automating different functionality. In particular, these view matchers are specifically designed for bidirectional localization issues (i.e. mirroring awareness). Furthermore, as a result of the translation, there are also some view assertions for cosmetic checks that need to be implemented as shown in Figure 13.2.

13.2 Testing Methods Implementation

In this section, we propose testing methods that can be used to test critical issues of bidirectional mobile software in general and specifically Arabic software.

13.2.1 Localized Resources Testing

Localized content includes text, style, layout and graphic artifacts on the user interface. Textual content includes the static text on UI elements like menus, buttons, text views, toast messages etc. For localization, everything needs to be translated in the respective language. This test verifies that all the required localized resources (files and folders) are present in the BiDi-language version and the localized content in each language that the product supports are loaded and retrieved successfully.

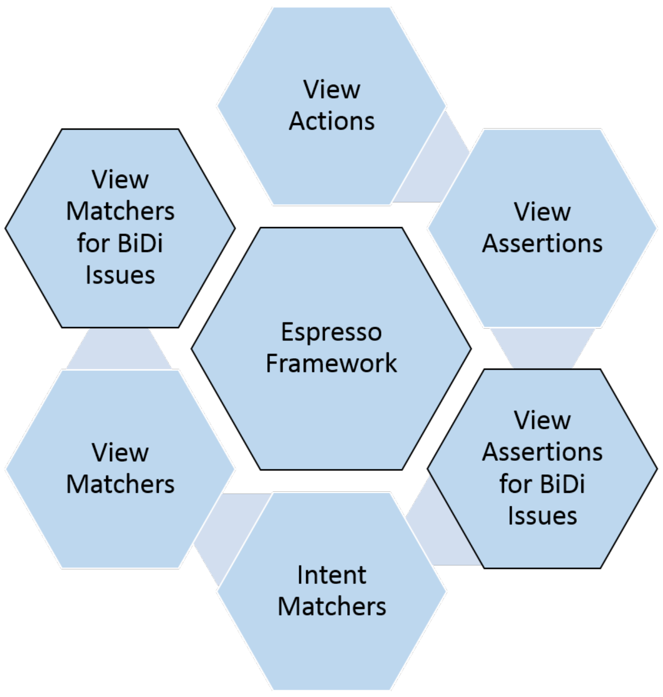


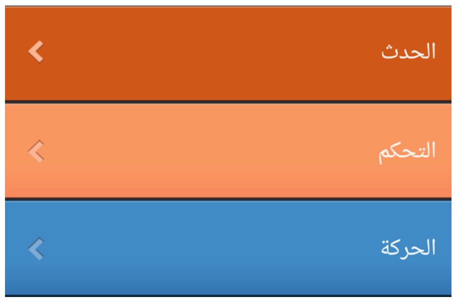
Figure 13.2: The customized version for Espresso API.

Localized strings validation

The user-facing text needs to be localized. Thus, the text needs to be translated to a specific language before it is displayed to the user. More specifically, all the strings should be stored in values resource files (see Figure 13.3(a)); the text can be retrieved by the software at the runtime to supply these elements to the users in the Arabic language as shown in Figure 13.3 (b). Hence, an alternative XML strings file must be created.

```
<string name="category_event">الحدث</string>
<string name="category_control">التحكم</string>
<string name="category_motion">الحركة</string>
```

(a)



(b)

Figure 13.3: Localized strings for Arabic language (a) String items (b) Screenshot for category's menu.

This test case should verify that the displayed texts match what are stored in values-ar resource folder for the Arabic language, and the text is retrieved correctly. Therefore, we need to implement a test case that capable of checking if the user-facing text matches the target translation which stored in the XML string file or not, and to verify that the string items values are loaded correctly for each UI element as shown in the below snippet of code.

```
@Test
  public void assertExistencesOfDefaultStringFile() {
      Context mContext = getActivity().getBaseContext();
      Resources res = getActivity().getResources();
      int checkExistenceForCategoryEvent =
          mContext.getResources().getIdentifier("category_event", "string",
          mContext.getPackageName());
      int checkExistenceForCategoryControl =
          mContext.getResources().getIdentifier("category_control", "string",
          mContext.getPackageName());
      int checkExistenceForCategoryMotion =
          mContext.getResources().getIdentifier("category_motion", "string",
          mContext.getPackageName());

      if (checkExistenceForCategoryEvent != 0 &&
          checkExistenceForCategoryControl != 0 &&
          checkExistenceForCategoryMotion != 0) { // the resource exists...
          result = true;
      } else { // checkExistenceForStrings == 0 // the resource does NOT exist!!
          result = false;
      }
      String failMsg = "The application default values resources are not exist";
      assertTrue(failMsg, result);
      String actualEventString = res.getString(checkExistenceForCategEvent);
      Espresso.onView(withText(R.string.category_event))
                              .check(matches(withText(actualEventString)));
      String actualControlString = res.getString(checkExistenceForCategControl);
      Espresso.onView(withText(R.string.category_control))
                              .check(matches(withText(actualControlString)));
      String actualMotionString = res.getString(checkExistenceForCategMotion);
      Espresso.onView(withText(R.string.category_motion))
                              .check(matches(withText(actualMotionString)));
  }
```

Fallback language

After the app is tested in all BiDi-languages and locales, additionally the fallback language of the app is checked. To test this, the device language is changed to one that is not supported and it is checked that the app uses the implemented fallback language. This test suite makes sure that the app runs properly and reverts to default resources. In the snippet of code below, the language configuration is changed to modern Chinese, which is not supported in the mobile app, and in that case the fallback language is checked to ensure that the app reverts to US English.

```
@Test
   public void assertFallBack() {
       setLocale(Locale.CHINA);
       onView(withText(R.string.app_name)).check(matches(withText("Pocket
           Code")));
   }
```

Application resources validation

It is tested whether alternative files of locale-specific resources, which are provided for the RTL mode, are retrieved correctly by the app at a runtime when the locale of the device is changed to the RTL language. Also, the test case is written to make sure that a full set of default resources regardless of language or locale is included in the app's structure as shown below.

```
 @Test
   public void assertDefaultResourcesAreExist() {
       Context mContext = getActivity().getBaseContext();
       int checkExistenceForDefaultStrings =
           mContext.getResources().getIdentifier("app_name", "string",
           mContext.getPackageName());
       int checkExistenceForDefaultDrawable =
           mContext.getResources().getIdentifier("icon_undo", "drawable",
           mContext.getPackageName());
       int checkExistenceForDefaultStyle =
           mContext.getResources().getIdentifier("MainMenuButton", "style",
           mContext.getPackageName());
       if (checkExistenceForDefaultStrings != 0 &&
           checkExistenceForDefaultDrawable != 0 &&
           checkExistenceForDefaultStyle != 0) { // the resources exist...
           result = true;
       } else { // checkExistenceForStyle == 0 // the resources does NOT exist!!
           result = false;
       }
       String failMsg = "The application resources are not exist";
       assertTrue(failMsg, result);
   }
```

13.2.2 Overtranslation

If the default texts are displayed instead of the translated texts in any part of the interface, then these texts may be overlooked or checked by reviewers. Such texts should not be translated and remain unmodified. In practice, the overtranslation testing is difficult to implement. However, the test should check that the text which is displayed in the text view is the same as the expected text defined in the XML strings file. Hence, it might be impractical to check all elements' text in the app.

Actually, in some situations, an untranslated string is not to be considered as a software defect. For example, the untranslated text might belong to a new feature added to the app that has not yet been translated, or to a UI element that should remain in the source language and is the same for all languages, such as a hyperlink or trademarked product names. Overtranslation issues are the responsibility of reviewers.

13.2.3 Linguistic Testing

Linguistic verification, or verifying the translation of all text on the localized product, is another very important task. In the proposed method, skilled linguistic experts can verify the linguistic content using screenshots of all the product screens to check the semantic correctness of the translations.

13.2.4 Cosmetic Testing

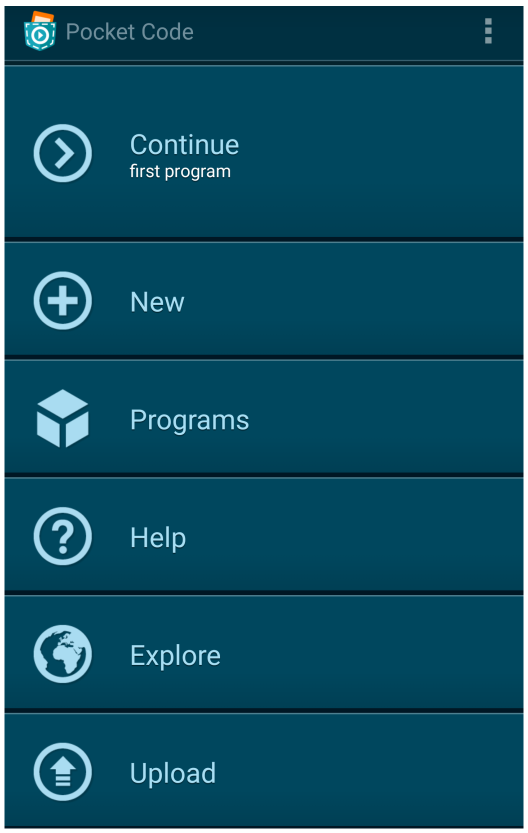
Translation has an enormous effect on the cosmetic quality of an app. The target of cosmetic testing is to verify that the localization phase did not introduce any visual defect and ensures that the UI has a consistent appearance throughout all supported BiDi-language versions, and a product contains no defects such as truncated strings, overlapping widgets, misaligned widgets, introduced during the localization process.

An important issue that needs to be addressed is the UI which contains layout, pop-up boxes, lists, and widgets. All UI elements need to support RTL direction as shown in Figure 13.4.

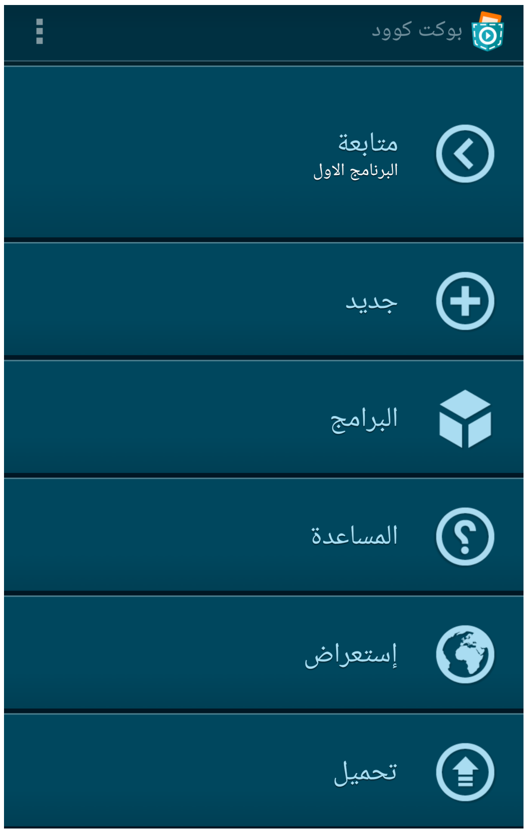
Mirroring awareness

For bidirectional languages, not only does the text alignment and text reading order render from right to left, but also the UI elements layout should follow this natural direction of rendering from right to left. To give a consistent right to left look and feel to an application's UI, both the text and the UI elements need to be laid out from right to left once they are switched to bidirectional languages. The UI of such languages is generally mirrored and right aligned. That is because those languages are written in a form known as RTL and strings flow in that direction.

In general, all UI elements should be mirrored, which includes lists, scrollbars, progress bars, pop-up boxes, grids, and galleries. The direction of text is also changed to RTL with the exception of phone numbers, country codes, and similar numbers which are by default LTR also in RTL languages. Note that the direction of some views and widgets in the UI layout may not change automatically. A new view matcher is proposed to check the mirroring awareness in the localized product. However, the LTR layout features after mirroring should match the BiDi features; otherwise, the test fails as shown in the below snippet of code.



(a)



(b)

Figure 13.4: Screenshot of the Pocket Code main menu (a) LTR (b) RTL.

```
public class MirroringMatcher {
   public static Matcher<Object> BrickLayoutIsMirrored(boolean isMirrored) {
      return LayoutShouldHaveMirroredValue(equalTo(isMirrored));
   }

   private static Matcher<Object> LayoutShouldHaveMirroredValue(final
       Matcher<Boolean> expectedObject) {
      final boolean[] isAutoMirrored = new boolean[1];
      return new BoundedMatcher<Object, BrickLayout>(BrickLayout.class) {
         @Override
         public boolean matchesSafely(final BrickLayout actualObject) {

            isAutoMirrored[0] = actualObject.getBackground().isAutoMirrored();

            if (expectedObject.matches(isAutoMirrored[0])) {
               return true;
            } else {
               return false;
            }
            }

            @Override
            public void describeTo(final Description description) {
                // Should be improved!
                description.appendText("Layout Mirroring did not match " +
                    isAutoMirrored[0]);
            }
        };
    }
}
```

A corresponding test case is implemented to check the mirroring awareness in the localized version as shown in the below snippet code.

```
Espresso.onView(withId(R.id.brick_broadcast_layout))
         .check(matches(MirroringMatcher.BrickLayoutIsMirrored(Boolean.TRUE)));
```

Direction-sensitive graphics

Naturally, in the RTL interface the time moves from right to left, and thus any "back" type arrow has to point to the right and the forward arrow to the left. However, the localization process changes the directions of the undo and redo icons to be easily understood by Arabic speakers. Equally important, the undo and redo icons should correctly express the direction of time in all languages. A new test case is proposed to check the mirroring awareness in localized product for direction-sensitive graphics using some image processing techniques. However, the LTR image features after mirroring should match the BiDi features; otherwise, the test fails (see Figure 13.5).

A corresponding test case is implemented to check the mirroring awareness in the localized version. However, the undo feature in RTL language is the same as a mirrored one in the original version. The two methods imageAreEquals() and doMirroring() are implemented to ensure the completeness of testing.

```
public void testDisableUndoRedoMirroring() {
       ImageButton undoRTL = (ImageButton) mSolo.getView(R.id.btn_top_undo);
       Bitmap RTLUndoBmp = ((BitmapDrawable) undoRTL.getDrawable()).getBitmap();
       mSolo.clickOnView(mButtonTopRedo);
       setLocale(Locale.ENGLISH);//Change the GUI to LTR
       ImageButton undoOriginal = (ImageButton) mSolo.getView(R.id.btn_top_undo);
       Bitmap originalUndoBmp = ((BitmapDrawable)
           undoOriginal.getDrawable()).getBitmap();
       Bitmap mirroredOriginalUndoBitmap = doMirroring(originalUndoBmp);
       assertTrue(imagesAreEquals(RTLUndoBmp, mirroredOriginalUndoBitmap));
   }
```

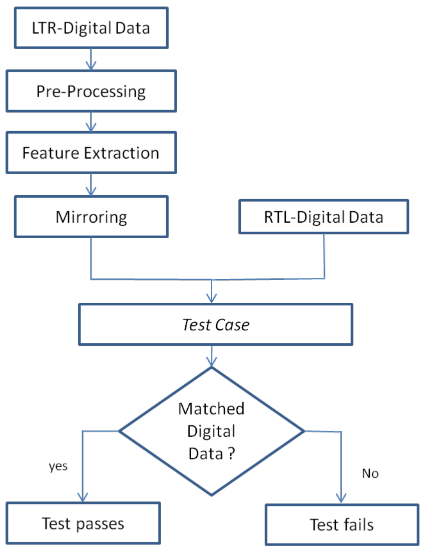


Figure 13.5: Flow chart for mirroring test-case.

Text direction

The direction of text becomes a critical issue when the app has Arabic texts, which are written and read from RTL. Hence, all user interface elements should support the Arabic language; the text alignment and text reading order go from RTL. The text direction of each view such as text view, edit text, button, and pop-up menu is tested. The testing method could be performed by supplying each user view with a text string from the Arabic language. The custom view assertion is created for Espresso framework, this view assertion is specifically designed to check the actual text direction for the views, a bug report is generated if the UI elements text direction is not right-to-left, after running with it throwing an exception if the condition is not met as shown in the code snippet below.

```
public class TextDirectionAssertions {
    public static ViewAssertion isTextDirectionRTL() {
        return new ViewAssertion() {
            public void check(View view, NoMatchingViewException noView) {
                assertThat(view, new
                    TextDirectionMatcher(View.TEXT_DIRECTION_RTL));
            }
        };
    }

    private static class TextDirectionMatcher extends BaseMatcher<View> {
        private int textDirection;

        public TextDirectionMatcher(int textDirection) {
            this.textDirection = textDirection;
        }

        @Override
        public void describeTo(Description description) {
            String TextDirection;
            if (textDirection == View.TEXT_DIRECTION_FIRST_STRONG) TextDirection
                = "Right to Left";
            else TextDirection = "Left to Right";
            description.appendText("View TextDirection must has equals " +
                TextDirection);
        }

        @Override
        public boolean matches(Object o) {
            if (o == null) {
                if (textDirection == View.TEXT_DIRECTION_FIRST_STRONG) return
                    true;
                else if (textDirection == View.TEXT_DIRECTION_LTR) return false;
            }
            if (!(o instanceof View))
                throw new IllegalArgumentException("Object must be instance of
                    View. Object is instance of " + o);
            return ((View) o).getTextDirection() == textDirection;
        }
    }
}
```

Completely displayed

Sometimes, translating string from one language to another changes the length of the string. The translated string length may be longer than the original one, which causes the string buffer to overflow (functional bug). So more space should be allocated than necessary for English. To accommodate most other languages up to 30 percent more is normal when translated into another language. This phenomenon is common for English abbreviation. For example, a computer terminology "GUI" which represents "Graphical User Interface", when translated into Arabic becomes "واجهة المستخدم الرسومية". In this case, the length of the string increases, and the string buffer may overflow.

Due to this issue, some views might be expanded and partially displayed on the screen, for instance there exist views (such as text view) whose height and width are larger than the physical device screen by design. Such views will never be completely displayed. If mobile tester wants to ensure the entire rectangle this view draws is completely displayed to the user and at least 100 percent of the view's area is displayed to the user, the completelyDisplayed method should be used as shown in the below Espresso snippet of code.

```
public static Matcher<View> isCompletelyDisplayed() {
    return isDisplayingAtLeast(100);
  }
```

The following snippet of code is written to inspect that a view whose height and width fit perfectly within the currently displayed region of this view:

```
Espresso.onView(withText("ControlString")).check(matches(isCompletelyDisplayed()));
```

String truncation validation

To test whether the UI is flexible enough to accommodate several language fonts and strings lengths, the following method is used to verify that a view is displayed correctly as it is defined in its XML layout file and that it fits into the UI without truncation. In Espresso libraries as shown below, the method is modified to assert that views hierarchy (e.g. text view, button, and spinner) do not contain ellipsized or cut off views.

```
 public static ViewAssertion noEllipsizedTextForTextView() {
        return selectedDescendantsMatch(
                isAssignableFrom(TextView.class), not(hasEllipsizedText()));
    }

    public static ViewAssertion noEllipsizedTextForButton() {
        return selectedDescendantsMatch(
                isAssignableFrom(Button.class), not(hasEllipsizedText()));
    }

    public static ViewAssertion noEllipsizedTextForSpinner() {
        return selectedDescendantsMatch(
                isAssignableFrom(Spinner.class), not(hasEllipsizedText()));
    }
```

A corresponding test case is implemented to verify that the text fits into the UI without truncation as shown in the below snippet code.

```
Espresso.onView(withText("ControlString")).check(noEllipsizedText());
```

Overlapping

After the translation is done, the localized tool can check the overlapping of widgets with other UI widgets. Mainly, the noOverlaps () method is used to ensure that the app you publish is devoid of overlapping in its UI. A bug report is generated if widgets are overlapped.

```
Espresso.onView(withText("ControlString")).check(LayoutAssertions.noOverlaps());
```

The Espresso noOverlaps method is modified to assert that descendant objects assignable to text view, image view, button, or edit text do not overlap each other as shown in the below snippet of code:

```
public static ViewAssertion noOverlaps() {
    return noOverlaps(allOf(
            withEffectiveVisibility(ViewMatchers.Visibility.VISIBLE),
            anyOf(
                    isAssignableFrom(TextView.class),
                    isAssignableFrom(ImageView.class)),
                    isAssignableFrom(Button.class),
                    isAssignableFrom(EditText.class)
    ));
}
```

Elements' positions

The localization process changes the UI layout due to the translation, and the translated text may affect the size of the UI elements. Therefore, some views' width and height usually increase from their original size. This size stretching may cause missing views. For this reason, the localized process should ensure that all the layout views are located on the screen as the UI design dictates. For this issue, the testOnScreen method is used, and an origin view is specified to start looking for the requested views.

In addition, the isLeftOf and isRightOf methods are used to check the positions of UI elements according to the locale direction and expose defects in the interface after mirroring. In practices, to localize the Brush tool button Figure 13.6 (a) to Arabic, the icon should be on the right side of text as shown in Figure 13.6 (c), not to the left as incorrectly shown in Figure 13.6(b).

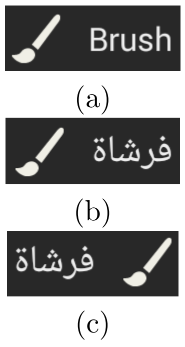


Figure 13.6: Screenshots for elements' positions in the tools dialog: (a) Original version (b) Incorrect position (c) Correct position .

The code snippet below illustrates the using of the isRightOf method to verify whether the icon is placed to the right of the text or not.

```
Espresso.onView(ViewMatchers.withId(RightControlID))
        .check(PositionAssertions.isRightOf(withId(LeftControlID)));
```

In Espresso framework, the (isRightOf and isLeftOf) methods are defined as shown in the

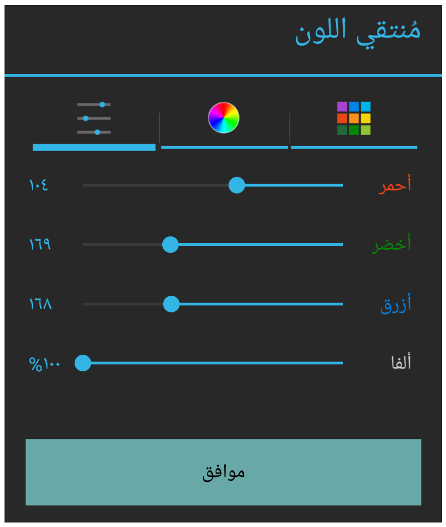


Figure 13.7: Screenshot of horizontal layout direction for RTL.

below snippet of code. Further, the methods throw an exception if there is more than 1 pixel of horizontal overlap:

```
public static ViewAssertion isRightOf(Matcher<View> matcher) {
  return relativePositionOf(matcher, Position.RIGHT_OF);
}
public static ViewAssertion isLeftOf(Matcher<View> matcher) {
  return relativePositionOf(matcher, Position.LEFT_OF);
}
```

Views swiping direction

Most widgets such as seek bars, progress bars, spinner, and outline views appear flipped. Each seek bar view is tested. The testing method could be performed by sliding the cursor of the seek bar to a new position. The horizontal layout direction of these views should be from RTL; otherwise, the view needs to be adjusted. The assert method is used to compare the expected layout direction results from the test to the actual layout direction as shown in Figure 13.7.

The following code snippet illustrates the details of the test case and how the horizontal layout direction of a seek bar is tested. The test case detects in which direction the user moved his or her finger when touching seek bar, whether the direction is to the left or right. Obviously, the test case simulates user behaviors and interactions. The two variables downXValue and upXValue are defined to store the values when the user's finger presses down and up, respectively. When the direction of the layout drawing is RTL, the test passes successfully.

```
public void testSeekBarDirection()
{
    mSolo.clickLongOnView(mRedSeekBar);
    mSolo.sleep(1000);
    mSolo.clickLongOnView(mRedSeekBar);
    mRedSeekBar.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {

        }
    });
    mRedSeekBar.setOnTouchListener(new View.OnTouchListener() {
        @Override
        public boolean onTouch(View v, MotionEvent event) {
            switch (event.getAction()) {
                case MotionEvent.ACTION_DOWN: {
            // store the X value when the user's finger was pressed down
                    downXValue = event.getX();
                    break;
                }
                case MotionEvent.ACTION_UP: {
            // Get the X value when the user released his/her finger
                    UpXValue = event.getX();
                    break;
                }
            }//end of Switch
            return true;
        }
    });
    mSolo.clickLongOnView(mRedSeekBar);
    mSolo.sleep(1000);
    mSolo.clickLongOnView(mRedSeekBar);
    mSolo.drag(mRedSeekBar.getLeft(),mRedSeekBar.getTop(),200,mRedSeekBar.getTop(),10);
    assertTrue(UpXValue>downXValue);
    String failMsg="The Direction of Red SeekBar is Left-to-Right";
    assertEquals(failMsg,mRedSeekBar.getLayoutDirection(),View.LAYOUT_DIRECTION_RTL);
    assertTrue(Integer.parseInt((String) mRedValueTextView.getText())>0);
}
```

Misalignment

The width of some views would change according to the length of the target translation strings (displayed strings). This causes the views to be incorrectly aligned when the string is changed from one language to another as the UI design dictates. User interface elements should be tested for adaptability to target language string. It should change accordingly to accommodate text with larger lengths without distorting the alignment.

Varying spacing requirements caused by translation. Some language specific text takes more space than others, so the interface should be able to adapt to this change without getting distorted. For Example, German text occupies more space to convey the same information in comparison to English.

This test compares the alignment of views in the RTL layout to the original one. It checks to ensure that if a set of views is aligned on a specific axis in the original layout, they are all aligned on the same axis. To examine the misalignment, the following test method is proposed.

The test case asserts that views are right aligned, that is, that their right edges are on the same x location. In order to verify that the views are aligned in the layout as we expect, the L10n tool should implement an isRightAlignedWih () method. For left alignment testing, the L10n tool needs to implement a test case to assert that views are left aligned.

It is also necessary for the localized tool to examine the correct positions for views before and after mirroring. Accordingly, the isBottomAlignedWith() method is used to assert that two views are bottom aligned, and their bottom edges are on the same y location before and after mirroring. In addition, a test case is introduced to verify that all views that are top aligned before mirroring are not adjusted after mirroring. If anything is found unsatisfactory, that UI element should be adjusted. Then these test cases should examine each activity, dialog, and other user interface layouts.

Auto layout in views

An effective and special testing case becomes essential for some languages with the longer idioms or special characters since these languages are the more error prone on the visual display. Obviously, widgets and dialogs must be set to be expandable, both horizontally and vertically, in order to accommodate variations in texts width and height. The fonts in the Arabic language can expand horizontally and vertically more than in the English language. The result may be a truncated text because the space provided by the UI widget (text view) is not enough. Some widgets have a fixed width. Thus, when the string length increases, it would cause a truncated text to be displayed, usually, indicates bad user experience.

The proposed method needs to check that all fixed origins, widths, and heights are eliminated in the app's views so that the localized text can reflow automatically when the language or locale is changed as shown in Figure 13.8 (compared to Figure 11.9). The test code asserts that all view should not be ellipsized, though may wrap, which means that the view can grow to become big enough to fit its own internal content.

Layout direction

The direction of layout becomes a critical issue when the app has bidirectional texts, which are written and read from RTL. The layout of all UI elements should follow this direction. Therefore supporting bidirectional scripts needs layout customizations for all user interface elements, including buttons, text views, seek bars, sliders, check boxes, menus, and dialog boxes.

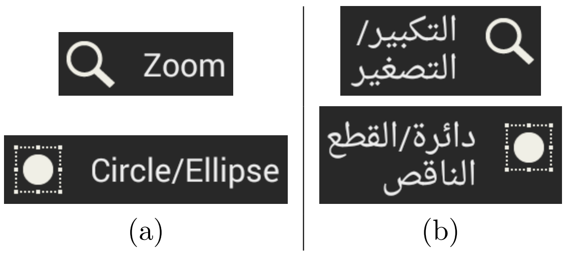


Figure 13.8: Screenshot of auto layout in Arabic version views.

The layout direction of each view is tested. The custom view assertion is created for Espresso framework, this view assertion is specifically designed to check the actual layout direction for the views, a bug report is generated if the UI elements layout direction is not right-to-left, after running with it throwing an exception if the condition is not met as shown in the code snippet below.

```java
public class LayoutDirectionAssertions {
    public static ViewAssertion isLayoutDirectionRTL() {
        return new ViewAssertion() {
            public void check(View view, NoMatchingViewException noView) {
                assertThat(view, new
                    LayoutDirectionMatcher(View.LAYOUT_DIRECTION_RTL));
            }
        };
    }

    private static class LayoutDirectionMatcher extends BaseMatcher<View> {
        private int layoutDirection;

        public LayoutDirectionMatcher(int layoutDirection) {
            this.layoutDirection = layoutDirection;
        }

        @Override
        public void describeTo(Description description) {
            String layoutDirectionStr;
            if (layoutDirection == View.LAYOUT_DIRECTION_RTL) {
                layoutDirectionStr = "Right to Left";
            } else {
                layoutDirectionStr = "Left to Right";
            }
            description.appendText("View LayoutDirection must has equals " +
                layoutDirectionStr);
        }

        @Override
        public boolean matches(Object object) {
            if (object == null) {
                if (layoutDirection == View.LAYOUT_DIRECTION_RTL) {
                    return true;
                } else if (layoutDirection == View.LAYOUT_DIRECTION_LTR) {
                    return false;
                }
            }

            if (!(object instanceof View)) {
                throw new IllegalArgumentException("Object must be instance of
                    View. Object is instance of " + object);
            }
            return ((View) object).getLayoutDirection() == layoutDirection;
        }
    }
}
```

Complex scripts awareness

Bidirectional text such as Arabic requires special rendering since there are different characteristics of scripts rendering include bi-direction, shaping as per context, reordering and linking characters. This testing describes the support provided by the app for international fonts, international text, and fine typography, so the app should display international text and render complex scripts, including bidirectional languages characters, contextual shaping, and ligatures.

The Optical Character Recognition (OCR) and text recognition approach retrieve the text of the control elements that are visible on the screen of the mobile device. To decide if the text is visible on the screen, OCR technology is used. OCR and text recognition tool can support various screen resolutions, orientations, and sizes. However, such tools can check only text elements that are visible on the screen. If the text changes or is eliminated from the app, the UI element is very hard (or impossible) to recognize. Another shortcoming of OCR recognition tools is that they are very slow as the whole screen needs to be scanned for the text.

However, the test scripts perform OCR to extract texts from screens as shown in Figure 13.9. Further, the test case should make sure that the app displays text using the fonts that are appropriate for the local market.

13.2.5 Functional Testing

Localization functional testing focuses on whether the localization phase introduced any problems and verifies that all features work as expected after the localization. Hence, all the functional test cases are executed after the localization process. For instance, test cases are created to check the formula editor functions in the localized version. However, such test scripts should make sure that the order in which the operators in an expression are evaluated (when the expression has several operators) adheres the precedence rules in mathematics and the result of the mathematical expression in the localized version equals the result in the original.

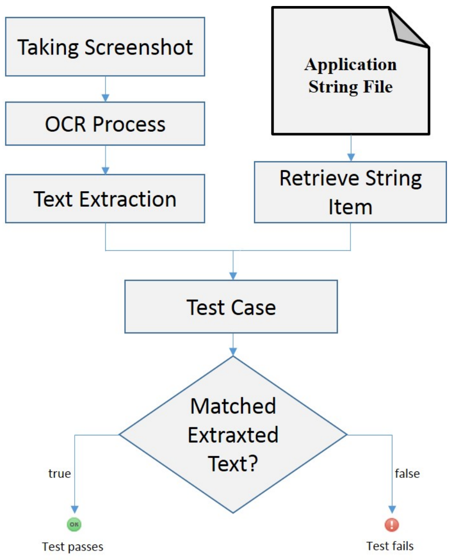


Figure 13.9: Flow chart for complex scripts awareness test-case.

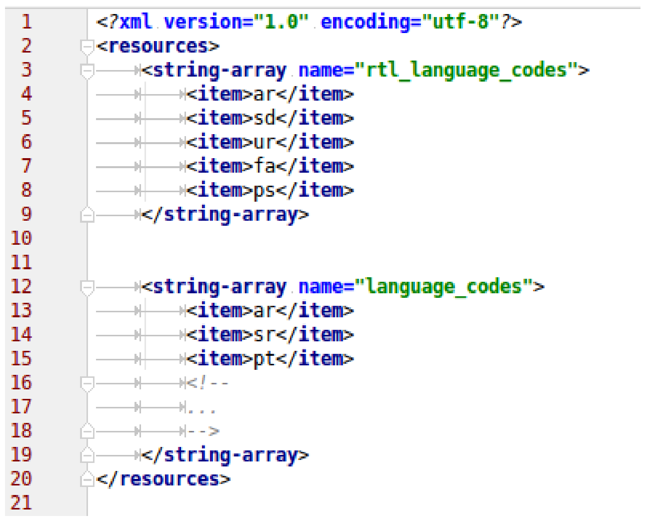
13.3 Providing Feedback About Rendering Correctness of Translated Languages

Another aspect of multi-language support is to ensure that translated strings are rendered correctly. Due to translations, the UI layout may slightly change, especially, when the translated text has a different length as the original hence more space will be allocated. This has an impact on the size of the UI elements. This change in size may cause clipped views, exceeding desired areas, or UI elements overlapping. For this reason, we need to ensure that all the layout views are drawn on the screen as the UI design dictates. This is especially important for mobile devices, if there are line breaks in the translation or parts of the strings are clipped or the position of the string near the icons does not meet the locale specification. The more information about rendering issues can be conveyed to the parties involved with fixing rendering/translation issues the quicker the job can be done.

A set of tests can check for correct rendering of the GUI elements of the current language. These tests must be run for all languages which are supported by the app. Since Pocket Code and Pocket Paint respectively support only a subset of languages of the Android OS or languages which are not yet supported by Android, the test must be aware which languages are supported by the app. The application level multilingual feature enables to run the test set for all supported languages since the language can be chosen via menu or programmatically.

13.3.1 Getting Information About All Languages Supported by the App

In our projects, we use Gradle to build the application and also to run the different kinds of tests: unit tests, static code analysis, and GUI tests. For the multi-language tests,



```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="rtl_language_codes">
        <item>ar</item>
        <item>sd</item>
        <item>ur</item>
        <item>fa</item>
        <item>ps</item>
    </string-array>


    <string-array name="language_codes">
        <item>ar</item>
        <item>sr</item>
        <item>pt</item>
        <!--
        ...
        -->
    </string-array>
</resources>
```

Figure 13.10: List of languages which are supported by the app in an XML-file in the test package's resources.

we use the Espresso testing framework which provides good performance and options to simulate user input and options to verify GUI rendering.

With the following code, one is able to get the supported languages of the operating system only which does not necessarily match with the supported languages by the app.

```
Locale locale : Locale.getAvailableLocales()
```

To collect the information which languages are supported by the app a Gradle task was established to create a list of languages which are stored in the resource folder in the test package and can be used by the multi-language GUI test to run the test-suite language by language. This is done by traversing the application's resource directory (res) prior to test execution, visit every values-directory having a language name and locale in the directory name, e.g.,values-de, or values-ar. Furthermore, it is checked whether in this directory a string.xml file exists - if so, there exists a translation for that language hence this language is supported by our app. Therefore it is appended to the supported language list and written into the resource folder of the test-package into an XML-file (see Figure 13.10). This file is later used by the Espresso/JUnit test to parametrize the test-suite execution.

13.3.2 Using the List of Supported Languages in JUnit for Test Parameterization

To run the test class's methods for all supported languages, the test class must be parameterized. This can be achieved by changing the test runner annotation of the class from:

```
@RunWith(AndroidJUnit4.class) to @RunWith(Parameterized.class)
```

Then a static method must be written to provide the list of parameters - in our case the supported languages the test methods are executed with as illustrated in the code snippet below.

```
 @Parameterized.Parameters(name = "testrun: {index} | lang-code: {0}")
   public static List<String> differentLanguages() {
       rtlLanguagesCodes =
           Arrays.asList(InstrumentationRegistry.getContext().getResources()
               .getStringArray(R.array.rtl_languages_codes));
       return Arrays.asList(InstrumentationRegistry.getContext().getResources()
               .getStringArray(R.array.rtl_languages_codes));
   }
```

Furthermore, the test class's constructor must be changed to accept a parameter from the list of parameters. This parameter - in our example a language of the list of supported languages - is passed to the constructor and stored for further use, e.g., in a member variable for access by the test methods as shown in the below snippet of code.

```
public ViewsAssertionLocalizationTest(String languageCode) {
       super(MainActivity.class);
       this.languageCode = languageCode;
   }
```

13.3.3 A Typical Test to Check Rendering of Different Languages in GUI

In our apps (Pocket Code/Pocket Paint) a typical language test is concerned with the rendering of the GUI. The GUI is tested for every different language for completeness/-correctness of rendering. This means that no clipping of strings or widgets occurs and that no translated string is rendered with line-breaks which would waste space for the other widgets or where it is required to be rendered in a single line like in the original (base) language. It is also necessary to check the correct positioning of the string in relation to icons on the GUI. This is especially important when the app is capable of switching between LTR and RTL languages. An example Espresso test method can be seen in the below snippet of code, which tests the rendering of the tools' dialog in Figure 13.6. In this test, the location of the text in relation to the tool icon (here Brush icon) is tested either for the right side if the app's language is set to LTR or for the left side if the language is set to an RTL (e.g., Arabic) language.

```
@Test
   public void assertToolsRightOfTextForRtlOrLeftOfTextForLtrLanguages() {
       onView(withId(R.id.btn_bottom_tools))
               .perform(click());
       if (rtlLanguagesCodes.contains(languageCode)) {
           onView(allOf(withId(R.id.tool_button_image), hasSibling(
                   withText("Brush"))))
                   .check(isRightOf(allOf(withId(R.id.tool_button_text),
                    hasSibling(
                       withText("Brush")))));
      } else {
         onView(allOf(withId(R.id.tool_button_image), hasSibling(
               withText("Brush"))))
               .check(isLeftOf(allOf(withId(R.id.tool_button_text),
                  hasSibling(
                     withText("Brush")))));
      }
   }
```

13.3.4 Being Aware of RTL Languages

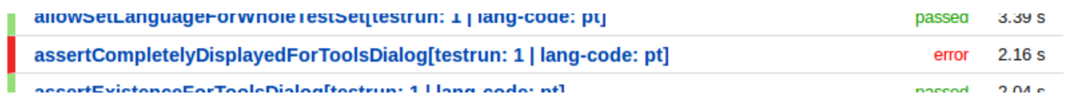
Usually, it is not a good testing practice to change a test's output depending on parameters. However, to get easily comparable sets of test methods we must react accordingly to a change from LTR to an RTL language and adapt those test methods which deal with a positioning of strings to either test for LTR or RTL language accordingly. Otherwise, it would be necessary to have a mapping of tests which are only aware of LTR and those which are only aware of RTL languages which would unnecessarily complicate the comparison of the overall test results between LTR and RTL languages.

13.3.5 Reacting to Failed GUI Rendering Tests

Assuming that the GUI rendering tests for the base language (English) are all green (i.e. no rendering issues) any failing tests for any of the supported languages must be processed properly which means that the failing tests and log-output are communicated to the coordinator who is responsible for the translations. Currently, in this workflow, the coordinator contacts the translators of the language the test failed and provides detailed information. This includes the string which caused the test to fail, the reason (line break, clipping) and how to possibly fix it. To be able to provide screenshots for exactly the failed tests all the tests concerned with GUI rendering are enclosed by a try-catch clause (see the code snippet below). GUI tests for checking the rendering can be enclosed by a try-catch clause. If the test fails, a screenshot is taken and the exception is rethrown, so the testrunner can recognize the failing test.

```
try {
     onView(withId(R.id.gridview_tools_menu)).check(noOverlaps());
} catch (AssertionFailedError e) {
       takeScreenshot(classAndMethodName);
       throw e;
}
```

These screenshots along with the log-output of the Espresso tests provide meaningful information for both developer and translator. Which language caused the issue can be seen in the log-output (see Figure 13.11). The information which test class, which test



```
allowSetLanguageForWholeTestSet[testrun: 1 | lang-code: pt]            passed   3.39 s
assertCompletelyDisplayedForToolsDialog[testrun: 1 | lang-code: pt]    error    2.16 s
assertExistenceForToolsDialog[testrun: 1 | lang-code: pt]              passed   2.04 s
```

Figure 13.11: Espresso test output.

method and which language is also encoded into the filename of the screenshot. The Jenkins job which runs these tests collects the log-output along with the screenshots and sends them to the coordinator who is responsible to assign developers and/or translators for solving these GUI rendering defects.

13.4 Testing Methods and Discussions

Primarily, to verify whether the target VPE meets the localization requirements or not, we need to check the product in terms of complying with Material Design guidelines for bidirectionality. Furthermore, during the localization phase, translation may cause some GUI defects (e.g. truncated strings, overlapping controls, and misalignment). These defects should be considered during the testing process as illustrated in Table 13.1.

In this section, we provide essential, full development, and user acceptance testing methods for the bidirectional languages localization issues. The test methods are executed on the bidirectional languages which are used as the reference. The methods had been created to perform a round trip and visit every screen, dialog, and menu. The test cases methods take screenshots of the visited screens and extract text strings for analysis purposes. The efficient testing methods are proposed to check the following issues.

Testing Methods Description	***Result***
Localized strings validation (i.e. user-facing text should be bidirectional scripts)	passed
Mirroring awareness: UI elements lay out from RTL	passed
Text direction (text reading order go from RTL)	passed
Complex scripts awareness (app render and edit bidirectional scripts)	passed
Direction-sensitive graphics (such as undo and redo)	passed
String truncation validation	passed
UI overlapping validation	passed
Views alignment (i.e. right-aligned elements)	passed
Linguistic verification: check the semantic correctness of the translations using product's screenshots	passed
Images and colors: issues of cultural appropriateness	passed

Table 13.1: Bidirectional languages testing methods.

13.4.1 Cultural Appropriateness

All of the application's graphics should be culture-neutral, the testing methods are performed at two levels. Each individual view is tested and each screen or layout is tested as a whole. Furthermore, the feature that supports localization of the view is tested. Full testing is performed at both the individual view and the screen level. For usability testing, two representatives of each culture participate. The participants check each view to determine whether or not there are any peculiarities of that view that might be offensive or include any attributes that use the idioms of a specific culture. If anything is found unsatisfactory, that view should be customized using the localization tool to be satisfactory.

13.4.2 Bidirectionality and Character Reordering

Testing is performed at two levels. Each individual view is tested or each text view, edit text, button, screen or menu. Furthermore, the facility that supports localization of the views is tested. Essential testing could be performed by supplying each user view a variety of text strings from the targeted languages using the localization tool. In this test, each control must be fed with a variety of text strings from different languages. This testing should result in a variety of changes in the text directionality and content format for that view.

For usability testing, two representatives of each culture participate. The participants should check each view to determine whether or not there are any peculiarities of that view that might need customization. If anything is found unsatisfactory, that view should be customized using the localization tool. Then these participants inspect each screen, layout, menu, and other UI elements.

13.4.3 Input Method Editors (IME)

The testing is done by using a localization tool that simulates a keyboard keystroke to input text into UI elements, such as edit text or text view. For user usability testing, two representatives of each culture participate. The participants check the displayed text character in each view to determine whether or not characters are being rendered correctly (correct mapping) or junk data is being rendered, whether or not characters are ordered correctly. If anything is found unsatisfactory, that view should be customized using the localization tool. Then these participants should examine each screen, layout, menu, and other UI elements.

13.4.4 Mirroring Awareness

The issue of UI layout includes activities, toast, dialog boxes, menus, seek bars, spinner, toolbars, and graphics. Essential testing for UI layout is performed by using a localization tool that transforms the coordinate of one, a set of UI elements or full screen from left-to-right and right-to-left, i.e. mirroring. The logical relationship and the attribute of layout's direction are maintained before and after mirroring. Full testing is done at the individual view level, on a set of views, and on the screen as a whole.

For usability testing, two representatives of each culture participate. The participants check the layout and the text directionality of the view to determine whether or not the view is correctly mirrored, logical relationships among UI elements are maintained, and the text direction follows the view layout. If anything is found unsatisfactory, that view should be customized using the localization tool. Then these participants review each screen, menu, and other UI elements.

13.4.5 Direction-Sensitive Graphics

The issue of icons that communicate direction includes undo, redo, back, and forward. Essential testing for direction-sensitive graphics is performed by using a localization tool that transforms the coordinate of one, or a set of icons from left-to-right and right-to-left, i.e. mirroring. The logical relationship and the directional attribute is maintained before and after mirroring.

For usability testing, two representatives of each culture participate. The participants check icons that communicate direction to determine whether or not the icon is correctly mirrored. If anything is found unsatisfactory, that icon should be customized using the localization tool. Then these participants review each icon.

From Table 13.1, we can observe that Pocket Code complies with the design guidelines for bidirectionality and not only does the text alignment and text reading order render from right to left, but also the UI elements layout follow this natural direction.

13.5 Benefits of the Automated Bidirectional Localization Testing Approach

Test automation can introduce many benefits to mobile app testing cycles, first of all, it is less time consuming and enable mobile testers to release better apps with less effort. Test automation allows performing several kinds of testing efficiently and effectively since the automated tests can run fast and frequently and increase the coverage of tests. Many companies still use primarily manual tests since they do not know the proper way to integrate automated testing in their app development process.

Test automation usually has a beneficial impact on mobile software development in general and specifically on internationalization and localization testing [6][16]. It increases the quality of the app and cuts down the time for testing although it does not come for free. Test automation is an investment which pays off when software is delivered. Once automatic tests have been established they can be run frequently and hence support the development process by increasing the confidence in one's code by building a test harness

[103]. However, let us discuss the benefits to perform automated bidirectional testing for mobile applications:

1. Reliability: Automated tests are insofar more reliable than manual tests since they are organized in suites (sets of tests) which are reliably executed whenever the suite is started. This means a test cannot be omitted or executed in a wrong way, but which is possible if done manually. Whenever tests are to be executed more than about 10 times they are candidates for automation if technically possible [104]. For instance, in Pocket Code/Paint we need to run the same test cases concerned with GUI rendering for every supported language this would mean to execute our example test suite of X test methods 30 times. Manually this would be error prone and a waste of time.

2. Increases test coverage and volume: Automated tests enable testers to run them on many different mobile devices with little effort. Manual testing would be impossible, with an increasing number of test cases, the effort for manual execution becomes disproportionately higher in comparison to the automated variant [104]. When an app supports multiple languages on app level it is even possible to test languages which are not supported by the operating system. Automated tests, by means of parameterization, are able to run the same tests (suite) for all different supported languages which would be a very daunting task manually. For instance navigation through the GUI of an app alone needs the manual tester to know how to navigate by heart or to be able to read the language displayed. Furthermore, automated tests navigate as quickly as the system allows to navigate or makes navigation step by step obsolete since the activity under test can be opened directly for testing purpose programmatically.

3. Continuity: The existent test code carries the information what kinds of tests have already been created, what defects were discovered and fixed. The test code is an executable specification. The overall test performance is summarized via clear reports after each execution.

4. Running tests anytime and anywhere: Having a continuous integration system in place it enables testers to start a complete test run regardless where testers are located in the world. Furthermore, these tests on the CI-system can be started manually, scheduled or triggered by an event (e.g., change request on source code). Test reports are collected by the CI-system and can be read asynchronously for a defined time span.

5. Saves time and money: To ensure quality and avoid regression the tests have to be repeated often during development. Every time source code is modified and a new language is supported the internationalization test cases should be repeated. For each release, all supported languages should be tested functional and for correct rendering. Again, manually repeating these test cases is an expensive waste of time. Automated tests can be run over and over again at no additional costs and they are much faster than manual testing [105].

6. Find bugs early: Test automation helps to discover bugs in the early phase of app development which decreases expenses. Furthermore, they help to decrease the chance

of reintroducing the same or similar bugs and provide immediate feedback when changes broke functionality.

7. Increasing in confidence: When an app is thoroughly tested, developers will gain confidence in their own code and that the app behaves as expected. Furthermore, it prevents the reintroduction of defects which have been already fixed since there are proper tests in place which would discover these defects automatically.

Chapter 14

Conclusion and Future Work

14.1 Conclusion

In this work, we localized a VPE on smartphones into bidirectional languages and presented the challenging aspects of the bidirectional languages (specifically Arabic language) facing the software localization team. These challenges stem from the fact that these languages differ enormously in terms of their characters, contextual shaping, and directionality from western languages [71].

Our software localization process includes adapting Catrobat to be used in the education system in primary and secondary schools. That process means changing, for example, custom and standard layout directions, text rendering and locale formats to suit the new culture it is being localized to. Our localization process also includes changing the source code of Catrobat and in many cases localizing icons to suit the local users' needs and expectations.

However, we laid out what is necessary to thoroughly support internationalization, localization and bidirectional script support and their intricacies. Concretely for the Arabic translation, this means that text must be translated, the icons correctly positioned in relation to the text, and that some icons representing “real” objects, e.g., a floppy disk or a globe, are not mirrored. Others icons conveying the flow of time or direction, e.g., undo, redo, back and forward are mirrored. We showed how the mobile app Pocket Paint was redesigned by complying with Google's Material Design guidelines. The UX test showed that the design changes led to an improvement in terms of simplicity, conciseness, and appeal.

Furthermore, in this thesis, an automation approach for L10n testing is introduced to localize an Android app with a rich GUI for the Arabic language and culture. The Arabic version of the original product looks as if it had been developed in the Arabic user's home country. On the one hand, traditional testing approaches can be applied to localize any software from English to other Latin-based languages. On the other hand, localization of mobile apps for BiDi-languages are particularly challenging to test. The bidi-language software requires savvier testing methods and efforts to ensure efficient testing. The Arabic language requires mirroring awareness to suit the RTL reading order and to provide a consistent right-to-left look and feel to the app. Localization to BiDi-languages affects not only the layout of text and UI elements but also iconography.

The proposed methods make sure the UI elements and icons that communicate direction are mirrored correctly in RTL locales. Furthermore, the proposed approach fully and completely considers all BiDi-languages issues in order to effectively reveal all common localization issues in the app's design. The methods ensure that the localized software meets a local user's expectations in terms of language, features, and user experience in the traditional sense. In essence, the automation approach saves time and effort and increases accuracy and repeatability for localization testing and maintains the usability and portability of the app. Moreover, it leads to a significant decrease in effort and time spent for regression testing. Finally, the approach's principles can easily be applied to any smartphone platform.

The localization testing results show that the product meets bidirectional requirements and complies with bidirectionality design guidelines since it is cosmetically correct, linguistically accurate, and culturally appropriate.

14.2 Future Work

There are some correlated challenging aspects for localization of the major Asian languages (Chinese, Korean, and Japanese). In particular, the peculiarities of Japanese and Korean languages raise some issues during the translation process for Catrobat. These challenges stem from the fact that the verb comes last in such languages, as well as that the plural and singular are the same.

Our future work is to propose a comprehensive solution that is based on the idea of implementing all strings with the placeholder for the parameters. This would allow translators to better understand the context and decide at which place the parameters should appear in the localized string. However, the Android string resource system does not support parameters, nor do localization management platforms.

For usability testing, it is planning to start a sequence of UX tests with larger sets of people (using sample set estimation according to [106]) of our target user group (age 13-17) to incrementally improve the usability and stability of Pocket Paint and Pocket Code especially with a focus on multi-language and bi-directional script support. Finally, the underlying rationale for this work was to broaden Pocket Paint and Pocket Code's user base. Therefore we will monitor the development of the number of downloads in Google's Play Store to see whether the language support on app level leads to more people installing and using the application in the specific regions of the world.

Bibliography

[1] P. Smutný. Visual programming for smartphones. In *12th International Carpathian Control Conference (ICCC)*, pages 358–361, May 2011.

[2] H. Tsukamoto, Y. Takemura, Y. Oomori, I. Ikeda, H. Nagumo, A. Monden, and K. I. Matsumoto. Textual vs. visual programming languages in programming education for primary schoolchildren. In *IEEE Frontiers in Education Conference (FIE)*, pages 1–7, Oct 2016.

[3] W. Slany. A mobile visual programming system for Android smartphones and tablets. In *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 265–266, Sept 2012.

[4] W. Slany. Catrobat. http://developer.catrobat.org/, 2010-2017 TU Graz, Graz, Austria (accessed February 10, 2016).

[5] C. Williams, E. Alafghani, A. Daley, K. Gregory, and M. Rydzewski. Teaching programming concepts to elementary students. In *IEEE Frontiers in Education Conference (FIE)*, pages 1–9, Oct 2015.

[6] A. M Ayyal Awwad and W. Slany. Automated bidirectional languages localization testing for Android apps with rich GUI. *Mobile Information Systems*, April 2016.

[7] Google. Material design is a unified system that combines theory, resources, and tools for crafting digital experiences. material design website. https://material.io/, 2014 (accessed April 10, 2017).

[8] I. K. Villanes, E. A. B. Costa, and A. C. Dias-Neto. Automated mobile testing as a service (AM-TaaS). In *IEEE World Congress on Services*, pages 79–86, June 2015.

[9] IDC Research. Apple tops samsung in the fourth quarter to close out a roller coaster year for the smartphone market, according to idc. http://www.idc.com/getdoc.jsp?containerId=prUS42268917, 2017 (accessed March 7, 2017).

[10] X. Xia, D. Lo, F. Zhu, X. Wang, and B. Zhou. Software internationalization and localization: An industrial experience. In *18th International Conference on Engineering of Complex Computer Systems*, pages 222–231, July 2013.

[11] Appannie.com. Languages supported in Google play reviews. https://support.appannie.com/hc/en-us/articles/204207814-Languages-supported-in-Google-Play-reviews, 2015 (accessed February 2, 2016).

[12] H. Dhingra and T. Roy. Localization testing in mobile world. http://conference.qaiglobalservices.com/stc2013/PDFs/Himanshu_Dhingra_2.pdf, 2013 (accessed July 10, 2015).

[13] C. Kopsch. Localization testing one-year status report for a localization project. *Testing Experience: Magazine for Professional Testers*, pages 20–22, Sep 2014.

[14] N. S. Cavalleri. Localization testing is more than testing the translation. *Testing Experience: Magazine for Professional Testers*, pages 23–23, Sep 2014.

[15] J. Gao, X. Bai, W. T. Tsai, and T. Uehara. Mobile application testing: A tutorial. *IEEE*, 47(2):46–55, Feb 2014.

[16] S. Abufardeh and K. Magel. Software internationalization: Testing methods for bidirectional software. In *Fifth International Joint Conference on INC, IMS and IDC*, pages 226–231, Aug 2009.

[17] S. Abufardeh and K. Magel. Software localization: The challenging aspects of Arabic to the localization process (Arabization). In *Proceedings of the IASTED International Conference on Software Engineering*, SE '08, pages 275–279, Feb 2008.

[18] C. A. Sakhr. Web-based Arabic-english mt engine. In *Proceedings of the Arabic NLP Workshop at ACL/EACL*, Toulouse, France, 2001.

[19] N. Kotze. Internationalization and localization testing. *Testing Experience: Magazine for Professional Testers*, pages 16–19, Sep 2014.

[20] C. Zhao, Z. He, and W. Zeng. Study on international software localization testing. In *Second World Congress on Software Engineering*, volume 2, pages 257–260, Dec 2010.

[21] Microsoft.com. Software internationalization. https://msdn.microsoft.com/en-us/globalization/software-internationalization.aspx, 2017 (accessed April 2, 2017).

[22] Net-Translators.com. Localization for mobile apps. https://www.net-translators.com/blog/localization-for-mobile-apps, 2015 (accessed January 2, 2016).

[23] Wordpress.com. Pseudo-localization testing in android. https://androidbycode.wordpress.com/tag/rtl/, 2015 (accessed January 20, 2016).

[24] R. Selvam and V. Karthikeyani. Mobile software testing- automated test case design strategies. *International Journal on Computer Science and Engineering*, 3(4):1450–1461, April 2011.

[25] M. Geogy and A. Dharani. Customising Android automated testing framework to enable native hardware and software support. *International Journal of Engineering Research and Technology*, 2(2):1–4, Feb 2013.

[26] K. Beck and C. Andres. *Extreme Programming Explained: Embrace Change (2nd Edition)*. Addison-Wesley Professional, 2004.

[27] D. Knott. *Hands-On Mobile App Testing: A Guide for Mobile Testers and Anyone Involved in the Mobile App Business (1st Edition)*. Addison-Wesley Professional, 2015.

[28] M. Sarwar and T. Rahim Soomro. Impact of smartphone's on society. *European Journal of Scientific Research*, 98(2):216–226, March 2013.

[29] B. Reed. A brief history of smartphone's. http://www.networkworld.com/slideshows/2010/061510-smartphone-history.html#slide, 2010 (accessed April 20, 2015).

[30] Statista.com. Smartphone shipments by vendor worldwide from 4th quarter 2009 to 4th quarter 2016 (in million units). https://www.statista.com/statistics/271490/quarterly-global-smartphone-shipments-by-vendor/, 2016 (accessed January 11, 2017).

[31] International Data Corporation. Smartphone OS market share, 2017 first quarter. http://www.idc.com/promo/smartphone-market-share/vendor, 2017 (accessed July 26, 2017).

[32] R. Islam, R. Islam, and T. Mazumder. Mobile application and its global impact. *International Journal of Engineering and Technology (IJEST)*, 10(6):72–78, Dec 2010.

[33] M. Kumar. Impact of the evolution of smart phones in education technology and its application in technical and professional studies: Indian perspective. *International Journal of Managing Information Technology*, 3(3):39–49, Sep 2011.

[34] P. Krebs and D. Duncan. Health app use among us mobile phone owners: A national survey. http://mhealth.jmir.org/2015/4/e101/, 2015 (accessed June 15, 2016).

[35] Itnonline.com. Two-thirds of Americans in favor of digital personal health management. https://www.itnonline.com/content/two-thirds-americans-favor-digital-personal-health-management, 2015 (accessed July 20, 2016).

[36] S. Misra. New report finds more than 165,000 mobile health apps now available. http://www.imedicalapps.com/2015/09/ims-health-apps-report/, 2015 (accessed July 22, 2016).

[37] S. Verstockt, D. Decoo, D. Van Nieuwenhuyse, F. De Pauw, and R. Van de Walle. Assistive smartphone for people with special needs : The personal social assistant. In *2nd Conference on Human System Interactions*, pages 331–337, May 2009.

[38] Statista.com. Number of apps available in leading app stores as of march 2017. https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/, 2017 (accessed April 17, 2017).

[39] Statista.com. Cumulative number of apps downloaded from the Google Play as of may 2016 (in billions). https://www.statista.com/statistics/281106/number-of-android-app-downloads-from-google-play/, 2017 (accessed April 17, 2017).

[40] Statista.com. Cumulative number of apps downloaded from the Apple app store from july 2008 to september 2016 (in billions). https://www.statista.com/statistics/263794/number-of-downloads-from-the-apple-app-store/, 2017 (accessed April 17, 2017).

[41] J. F. DiMarzio. *Android : A Programmer's Guide(1st Edition)*. McGraw-Hill, 2008.

[42] P. J. Deitel and H. Deitel. *Android How to Program(3rd Edition)*. Pearson Education, 2016.

[43] Android. Open handset alliance. https://www.openhandsetalliance.com/, accessed March 2, 2017.

[44] Android. Dashboards-platform versions. https://developer.android.com/about/dashboards/index.htmlPlatform, 2017 (accessed July 17, 2017).

[45] Android. Android studio. https://developer.android.com/studio/index.html, 2017 (accessed March 13, 2017).

[46] Android. Platform architecture. https://developer.android.com/guide/platform/index.html, 2017 (accessed March 21, 2017).

[47] C. Hu and I. Neamtiu. Automating GUI testing for Android applications. In *Proceedings of the 6th International Workshop on Automation of Software Test*, AST '11, pages 77–83. ACM, May 2011.

[48] Android. Activities. https://developer.android.com/guide/components/activities/index.html, 2017 (accessed March 21, 2017).

[49] Android. The activity lifecycle. https://developer.android.com/guide/components/activities/activity-lifecycle.html, 2017 (accessed March 21, 2017).

[50] Android. Publish your app. https://developer.android.com/studio/publish/ index.html, 2017 (accessed January 19, 2017).

[51] Android. Launch checklist. https://developer.android.com/distribute/best-practices/launch/launch-checklist.html, 2017 (accessed January 19, 2017).

[52] Apple. App store. https://developer.apple.com/support/app-store/, 2017 (accessed January 19, 2017).

[53] Microsoft.com. Publish Windows apps. https://developer.microsoft.com/en-us/store/publish-apps, 2017 (accessed February 10, 2017).

[54] H. Muccini, A. Di Francesco, and P. Esposito. Software testing of mobile applications: Challenges and future research directions. In *7th International Workshop on Automation of Software Test (AST)*, pages 29–35, June 2012.

[55] B. Kirubakaran and V. Karthikeyani. Mobile application testing challenges and solution approach through automation. In *International Conference on Pattern Recognition, Informatics and Mobile Engineering*, pages 79–84, Feb 2013.

[56] OpenSignal.com. Android fragmentation visualized. https://opensignal.com/reports/2015/08/android-fragmentation/, 2015 (accessed November 26, 2016).

[57] E. Hau and M. Aparício. Software internationalization and localization in web based ERP. In *Proceedings of the 26th Annual ACM International Conference on Design of Communication*, SIGDOC '08, pages 175–180. ACM, Sep 2008.

[58] Microsoft.com. Software internationalization. https://msdn.microsoft.com/en-US/globalization/mt642951, 2017 (accessed January 20, 2017).

[59] A. M. Ayyal Awwad. Localization to bidirectional languages for a visual programming environment on smartphones. *International Journal of Computer Science Issues*, 14(3), May 2017.

[60] S. Gross. Internationalization and localization of software. Master's thesis, Eastern Michigan University, 2006.

[61] Statista.com. Share of app users who have stopped using an app because it was not localized properly as of march 2014. https://www.statista.com/statistics/296304/mobile-app-abandoment-rate-due-to-lacking-localization/, 2014 (accessed January 5, 2017).

[62] R. Weber. App store optimization. http://nativex.com/wp-content/uploads/2016/02/NativeX_ASO_Whitepaper.pdf, 2016 (accessed February 10, 2017).

[63] Statista.com. Smartphone user penetration as percentage of total population in Middle East & Africa from 2011 to 2018. https://www.statista.com/statistics/203732/smartphone-penetration-per-capita-in-the-middle-east-and-africa-since-2005/, 2014 (accessed February 11, 2017).

[64] Statista.com. Number of smartphone users in Saudi Arabia from 2014 to 2021 (in millions). https://www.statista.com/statistics/494616/smartphone-users-in-saudi-arabia/, 2015-2016 (accessed February 10, 2017).

[65] Statista.com. Smartphones, tablets spread across the Middle East and Africa. https://www.emarketer.com/Article/Smartphones-Tablets-Spread-Across-Middle-East-Africa/1012989, 2015 (accessed March 15, 2017).

[66] WHP International SAS. Challenges of Arabic localization. http://www.whp.net/wp-content/uploads/2015/07/150721_Challenges-of-Arabic-Localization.pdf, 2015 (accessed December 20, 2016).

[67] International Telecommunication Union. 75% of online Arabic content is of Jordanian origin. http://www.bi-me.com/main.php?id=57309t=1, 2012 (accessed April 10, 2016).

[68] Internet World Stats. Internet world users by language. http://www.internetworldstats.com/stats7.htm, 2016 (accessed February 2, 2017).

[69] Alarabiya.net. Google: Arabic content ranks eighth on the internet. http://english.alarabiya.net/en/media/digital/2013/12/01/Google-Arabic-content-ranks-eighth-on-the-internet.html, 2013 (accessed May 12, 2016).

[70] A. Alqudsi, N. Omar, and K. Shaker. Arabic machine translation: A survey. *Artificial Intelligence Review*, 42(4):549–572, Dec 2014.

[71] T. A. El-Sadany and M. A. Hashish. An Arabic morphological system. *IBM Systems Journal*, 28(4):600–612, Nov 1989.

[72] W. Slany. Catrobat. http://no1leftbehind.eu/pocket-code/, 2010-2017 TU Graz, Graz, Austria (accessed February 10, 2017).

[73] W. Slany. Pocket Code: A Scratch-like integrated development environment for your phone. In *Proceedings of the Companion Publication of the 2014 ACM SIGPLAN Conference on Systems, Programming, and Applications: Software for Humanity*, SPLASH '14, pages 35–36. ACM, Oct 2014.

[74] W. Slany. Catrobat education. https://edu.catrob.at/brick-documentation, 2010-2017 TU Graz, Graz, Austria (accessed February 1, 2017).

[75] A. M. Ayyal Awwad, C. Schindler, K. Kumar Luhana, Z. Ali, and B. Spieler. Improving Pocket Paint usability via Material Design compliance and internationalization & localization support on application level. In *Proceedings of the 19th International Conference on Human-Computer Interaction with Mobile Devices and Services*. ACM, Sep 2017.

[76] M. Kropp and P. Morales. Automated GUI testing on the Android platform. In *Proceedings of the 22nd IFIP International Conferenceon Testing Software and Systems*, pages 67–72, Oct 2010.

[77] A. Kumar Maji, K. Hao, S. Sultana, and S. Bagchi. Characterizing failures in mobile OSes: A case study with Android and Symbian. In *IEEE 21st International Symposium on Software Reliability Engineering*, pages 249–258. IEEE, Nov 2010.

[78] V. Garousi and M. V. Mäntylä. When and what to automate in software testing? a multi-vocal literature review. *Information and Software Technology*, 76(C):92–117, Aug 2016.

[79] J. Gao, X. Bai, W. T. Tsai, and T. Uehara. Mobile application testing: A tutorial. *IEEE*, 47(2):46–55, Feb 2014.

[80] I. Burnstein. *Practical Software Testing: A Process-Oriented Approach.* Springer, 2003.

[81] A. Ben David. Mobile app testing. *Testing Experience: Magazine for Professional Testers*, pages 64–65, Sep 2012.

[82] R. Fernández Álvarez. Testing on a real handset vs. testing on a simulator–the big battle. *Testing Experience: Magazine for Professional Testers*, pages 42–44, Sep 2012.

[83] K. Rayachoti. Mobile test strategy. *Testing Experience: Magazine for Professional Testers*, pages 30–33, Sep 2012.

[84] H. Zadgaonkar. *Robotium Automated Testing for Android.* Packt Publishing, 2013.

[85] Google. Espresso. https://google.github.io/android-testing-support-library/docs/espresso/index.html, 2017 (accessed February 20, 2016).

[86] P. Blundell and D. Torres Milano. *Learning Android Application Testing.* Packt Publishing, 2015.

[87] M. Soni. *Jenkins Essentials.* Packt Publishing, 2015.

[88] S. Chacon. *Pro Git.* Apress (1st ed), 2009.

[89] T. Berglund and M. McCullough. *Building and Testing with Gradle: Understanding Next-Generation Builds.* O'Reilly Media(1st ed), 2011.

[90] R. Meier. *Professional Android 2 Application Development.* Wrox (2nd ed), 2010.

[91] Android Developers. Locale. http://developer.android.com/reference/java/util/ Locale.html, 2016 (accessed January 2, 2016).

[92] S. Abufardeh and K. Magel. QA/testing bi-directional languages software: Issues and challenges. In *Annual IEEE International Computer Software and Applications Conference*, pages 172–175, July 2008.

[93] Android Developers. Create resizable bitmaps (9-patch files). https://developer.android.com/studio/write/draw9patch.html, 2016 (accessed December 10, 2016).

[94] Android Developers. View. https://developer.android.com/reference/android/view/ View.html, 2017 (accessed January 21, 2017).

[95] B. DiSalvo. Graphical qualities of educational technology: Using drag-and-drop and text-based programs for introductory computer science. *IEEE Computer Graphics and Applications*, 34(6):12–15, Nov 2014.

[96] OpenGL. Texture mapping. https://www.opengl.org/archives/resources/faq/ technical/texture.htm, 2017 (accessed February 2, 2017).

[97] F. Lachner, P. Naegelein, R. Kowalski, M. Spann, and A Butz. Quantified UX: Towards a common organizational understanding of user experience. In *Proceedings of the 9th Nordic Conference on Human-Computer Interaction*, NordiCHI '16, pages 1–10. ACM, Oct 2016.

[98] Google. Material Design is a unified system that combines theory, resources, and tools for crafting digital experiences. https://material.io, 2014 (accessed April 1, 2017).

[99] I. P. Kuzheleva-Sagan and N. A. Suchkova. Designing trust in the Internet services. *AI Soc.*, 31(3):381–392, Aug 2016.

[100] W. Shu-Yi and F. Kuo-Kuang. Survey into user's usage feeling towards material design button in the website. In *International Conference on Advanced Materials for Science and Engineering (ICAMSE)*, pages 632–635, Nov 2016.

[101] Statista.com. Forecasted unit shipments of smartphones worldwide from 2016 to 2021 (in million units), by operating system. https://www.statista.com/statistics/309448/global-smartphone-shipments-forecast-operating-system, 2016 (accessed April 10, 2017).

[102] J. Nielsen. Estimating the number of subjects needed for a thinking aloud test. *International journal of human-computer studies*, 41(3):385–397, Sep 1994.

[103] D. Mohammad Rafi, K. Reddy Kiran Moses, K. Petersen, and M. V. Mäntylä. Benefits and limitations of automated software testing: Systematic literature review and practitioner survey. In *7th International Workshop on Automation of Software Test (AST)*, pages 36–42, June 2012.

[104] S. Berner, R. Weber, and R. K. Keller. Observations and lessons learned from automated testing. In *Proceedings. 27th International Conference on Software Engineering, 2005. ICSE 2005.*, pages 571–579, May 2005.

[105] R. Angmo and M. Sharma. Performance evaluation of web based automation testing tools. In *5th International Conference - Confluence The Next Generation Information Technology Summit*, pages 731–735, Sept 2014.

[106] C. W. Turner, J. R. Lewis, and J. Nielsen. Determining usability test sample size. In *In W. Karwowski (ed.), International encyclopedia of ergonomics and human factors*, pages 3084–3088, Jan 2006.