Certified Tester Specialist

ISTQB® Mobile Application Testing Foundation Level

Syllabus

Version 2019

Provided by International Software Quality Institute (iSQI)

International Software Testing Qualifications Board

This document was produced by a core team from the International Software Testing Qualifications Board Mobile Application Testing Working Group.

Vipul Kocher (chair), Piotr Wicherski (vice-chair), José Díaz, Matthias Hamburg, Eran Kinsbruner, Björn Lemke, Samuel Ouko, Tal Pe'er, Ralf Pichler, Lloyd Roden, Nils Röttger, Angelina Samaroo, Yaron Tsubery

Revision History

Version	Date	Remarks
Alpha	11 May 2018	Alpha Release
Beta	27 January 2019	Beta Release
GA	28 March 2019	GA Release
V2019	3 May 2019	ISTQB® Release

Table of Contents

Acknowledgments

This document was produced by a core team from the International Software Testing Qualifications Board Mobile Application Testing Working Group.

Vipul Kocher (chair), Piotr Wicherski (vice-chair), José Díaz, Matthias Hamburg, Eran Kinsbruner, Björn Lemke, Samuel Ouko, Tal Pe'er, Ralf Pichler, Lloyd Roden, Nils Röttger, Angelina Samaroo, Yaron Tsubery

The core team thanks the review team for their suggestions and input.

The following persons participated in the reviewing, commenting or balloting of this syllabus:

Graham Bath, Veronica Belcher, Armin Born, Geza Bujdoso, YongKang Chen, Wim Decoutere, Frans Dijkman, Florian Fieber, David Frei, Péter Földházi Jr., Chaonian Guo, Attila Gyuri, Ma Haixia, Matthias Hamburg, Zsolt Hargitai, Hongbiao Liu, Ine Lutterman, Marton Matyas, Petr Neugebauer, Ingvar Nordström, Francisca Cano Ortiz, Nishan Portoyan, Meile Posthuma, Emilie Potin-Suau, Liang Ren, Lloyd Roden, Chaobo Shang, Mike Smith, Péter Sótér, Marco Sogliani, Michael Stahl, Chris Van Bael, Paul Weymouth, Salinda Wickramasinghe, Minghui Xu

This document was formally released by the ISTQB® on 3 May 2019.

0. Introduction

0.1 Purpose of this Document

This syllabus forms the basis for the qualification of Mobile Application Testing at the Foundation Level. The ISTQB® provides this syllabus as follows:

1. To National Boards, to translate into their local language and to accredit training providers. National Boards may adapt the syllabus to their particular language needs and modify the references to adapt to their local publications.
2. To Exam Boards, to derive examination questions in their local language adapted to the learning objectives for each syllabus.
3. To training providers, to produce courseware and determine appropriate teaching methods.
4. To certification candidates, to prepare for the exam (as part of a training course or independently).
5. To the international software and systems engineering community, to advance the profession of software and systems testing, and as a basis for books and articles.

The ISTQB® may allow other entities to use this syllabus for other purposes, provided they seek and obtain prior written permission.

0.2 The Certified Foundation Level Mobile Application Testing

The Foundation Level qualification is aimed at anyone involved in software testing who wishes to broaden their knowledge of Mobile Application Testing or anyone who wishes to start a specialist career in Mobile Application Testing.

Information about Mobile Application Testing described in the ISTQB® Certified Tester Foundation Level syllabus [ISTQB_CTFL_2018] has been considered in creating this syllabus.

0.3 Business Outcomes

This section lists the business outcomes expected of a candidate who has achieved the Foundation Level Mobile Application Testing certification.

MAT-01 Understand and review business and technology drivers for mobile apps in order to create a test strategy.

MAT-02 Identify and understand the key challenges, risks and expectations associated with testing a mobile application.

MAT-03 Apply test types and levels specific to mobile applications.

MAT-04 Apply common test types, such as those mentioned in [ISTQB_CTFL_2018], in the mobile specific context.

MAT-05 Carry out the activities required specifically for mobile application testing as part of the main activities described in the ISTQB® test process.

MAT-06 Identify and use suitable environments and appropriate tools for mobile application testing.

MAT-07 Understand methods and tools used specifically to support mobile application test automation.

0.4 Examinable Learning Objectives

These learning objectives support the business outcomes and are used to create the examination which must be passed to achieve the Foundation Level Mobile Application Testing Certification. Learning objectives are allocated to a cognitive level of knowledge (K-Level).

A K-level, or cognitive level, is used to classify learning objectives according to the revised taxonomy from Bloom [Anderson 2001]. ISTQB® uses this taxonomy to design examinations against its syllabi.

This syllabus considers three different K-levels (K1 to K3). For more information see chapter 7.

0.5 Hands-on Levels of Competency

Mobile Application Testing Foundation Level introduces the concept of Hands-On Objectives which focus on practical skills and competencies.

Competencies can be achieved by performing hands-on exercises, such as those shown in the following non-exhaustive list:

- Exercises for K3 level learning objectives performed using paper and pen or word processing software, as is done for various existing ISTQB® syllabi.
- Setting up and using test environments.
- Testing applications on virtual and physical devices.
- Using tools on desktops and/or mobile devices to test or assist in testing related tasks such as installation, querying, logging, monitoring, taking screenshots etc.

The following levels apply to hands-on objectives:

- H0: This can include a live demo of an exercise or recorded video. Since this is not performed by the trainee, it is not strictly an exercise.
- H1: Guided exercise. The trainees follow a sequence of steps performed by the trainer.
- H2: Exercise with hints. The trainee is given an exercise with relevant hints to enable the exercise to be solved within the given timeframe.
- H3: Unguided exercises without hints.

Recommendations:

- K1 learning objectives typically use H0 level and H1 or H2 when the situation demands.
- K2 learning objectives typically use H1 or H2 levels and H0 or H3 when the situation demands.

K3 learning objectives typically use H2 or H3 levels, although it is not always necessary to have a hands-on exercise for a K3 learning objective. If the setup is complex or if it will be too time consuming, then use H0 level.

0.6 The Examination

The Mobile Application Testing Foundation Level Certificate exam will be based on this syllabus. Answers to exam questions may require knowledge based on more than one section of this syllabus. All sections of the syllabus are examinable, except for the introduction and appendices. Standards, books, and other ISTQB® syllabi are included as references, but their content is not examinable, beyond what is summarized in this syllabus itself.

The Mobile Application Testing Foundation Level exam is multiple choice. There are 40 questions. To pass the exam, at least 65% of the questions (i.e., 26 questions) must be answered correctly. Hands-on objectives and exercises will not be examined.

Exams may be taken as part of an accredited training course or taken independently (e.g., at an exam center or in a public exam). Completion of an accredited training course is not a prerequisite for the exam.

Somebody who wants to appear for the exam without getting trained from an accredited training provider should read the competence guidelines as provided in the Accreditation and Competence Guidelines document [CTFL-MAT-2019-Accreditation-and-Competence-Guidelines.pdf] and try to conduct these hands-on exercises themselves. This will help them gain the competencies that an accredited training provider would be expected to impart. Please note that this has no bearing on the Mobile Application Testing Foundation Level certification examination as the exam is only based on this syllabus and its learning objectives.

0.7 Recommended Training Times

A minimum training time has been defined for each learning objective in this syllabus. The total time for each chapter is indicated in the chapter heading.

Training providers should note that other ISTQB® syllabi apply a “standard time” approach which allocates fixed times according to the K-Level. The Mobile Application Testing syllabus does not strictly apply this scheme. As a result, training providers are given a more flexible and realistic indication of minimum training times for each learning objective.

0.8 Entry Requirements

The ISTQB® Foundation Level certificate must be obtained before taking this exam.

0.9 Sources of Information

Terms used in the syllabus are defined in the ISTQB®'s Glossary of Terms used in Software Testing [ISTQB_GLOSSARY].

Chapter 6 contains a list of recommended books and articles on Mobile Application Testing.

1. Mobile World - Business and Technology Drivers - 175 mins

Keywords

risk analysis, risk mitigation, risk-based testing, test strategy

Learning Objectives for Business and Technology Drivers

1.1 Mobile Analytics Data
MAT-1.1.1 (K2) Describe how available mobile analytics data can be used as input for the test strategy and the test plan.
HO-1.1.1 (H3) Based on data collected from one or more analytics data sources (geographical location, platform, operating system version and device type distribution), select the device types to be tested and their corresponding prioritization.
Note: HO-1.1.1 and HO-1.7.1 (below) may be combined.

1.2 Business Models for Mobile App
MAT-1.2.1 (K2) Distinguish between various business models for mobile applications.

1.3 Mobile Device Types
MAT-1.3.1 (K1) Recall different types of mobile devices.

1.4 Types of Mobile Applications
MAT-1.4.1 (K2) Distinguish between different types of mobile applications.

1.5 Mobile Application Architecture
MAT-1.5.1 (K2) Distinguish between general architecture types of mobile applications.

1.6 Test Strategy for Mobile Apps
MAT-1.6.1 (K3) Apply characteristics and specifics of the mobile market in preparing a test strategy.

1.7 Challenges of Mobile Application Testing
MAT-1.7.1 (K2) Give examples of the challenges associated with testing mobile applications.
HO-1.7.1 (H1) Gather market data such as device or operating system market share for a selected region. Gather data for screen sizes and density. Create a list of five devices and calculate the expected market coverage for this list.
Note: HO-1.1.1 (see earlier) and HO-1.7.1 may be combined.

1.8 Risks in Mobile Application Testing
MAT-1.8.1 (K2) Describe how risks specific to mobile applications may be mitigated.

1.1 Mobile Analytics Data

There are many stakeholders in the mobile world including manufacturers, platform providers, operating system (OS) providers, market data providers, tool providers and, of course, application developers and testers.

In order to contribute effectively to test planning discussions and test analysis, a mobile application tester should be aware of and familiar with the following factors:

- The business implications of the distribution of platforms

- Application downloads per platform
- The quantity and distribution of OS versions
- The market distribution of various device types, including variations based on geographical location
- Differing screen sizes and resolutions
- The various input methods
- Camera types

There are several sources of information for the above, both free and commercial-based. These include StatCounter GlobalStats [URL1], the OS vendors themselves and other third-party sources.

The mobile analytics data is used to select a device portfolio for test execution that is appropriate for the target market. Tests are run over this portfolio to test the app on a device in accordance with the importance of the device. The data related to devices and their special features, if any, may also be used to design tests specific to a device type. For example, a device with heart beat sensor may need special test cases.

1.2 Business Models for Mobile Apps

There are several models which can be used to monetize the work done in creating mobile applications. These include but are not limited to: Freemium, advertisement-based, transaction-based, fee-based, and enterprise applications. In addition, in-app purchases can be applied to some of these models.

There are certain advantages and disadvantages for each of these approaches and the tester should keep the business model in mind whilst testing the mobile application.

(1) In a Freemium model the applications are generally free but users have to pay if the need additional features. The applications need to provide sufficient features to be attractive to the users, whilst at the same time providing advanced features for which a large number of users would be willing to pay.

(2) Advertisement-based applications display advertisements on the screen as users interact with the applications. This strategy for revenue generation is more effective if the applications are used for relatively long periods of time. The user interface designers must take care when displaying the advertisements. They must be prominent enough without hiding essential parts of the application and they must ensure that users are not distracted and dislike using the application.

(3) Transaction-based applications charge the users either per transaction, a flat fee or a percentage of the transaction value or similar. This type of business model is suitable for a limited number of applications only and is usually applied for business and financial apps such as mobile wallets.

(4) Fee-based applications require the users to pay for downloading and installing the application. Deciding on a fee-based business model should be well-considered since large numbers of free or freemium options exist for most application types. The probability of users buying such an app increases if it provides outstanding features or usability, or when competing applications are not available.

(5) Free and enterprise applications do not charge their users. Enterprise applications are developed for internal use within the organization and provide an interface to the services provided. There are many such apps available from organizations such as banks or e-commerce companies. These apps are not generally focused on monetizing the app itself but allow revenue to be generated by directing users to the services provided by the organizations.

1.3 Mobile Device Types

There is a variety of mobile devices available that support different types of applications.

Typical devices include:

- Basic phones
- Feature phones
- Smartphones
- Tablets
- Companion devices - including wearables and some IoT (Internet of Things) devices.

When testing it should be kept in mind that each type of device has specific features for particular needs.

Basic phones are used for telephone and SMS only and provide very few built-in apps and games. The installation of apps or browsing is not possible.

Feature phones provide limited support for apps and app installation. They provide internet access via a built-in browser and may have some additional hardware such as cameras.

Smartphones provide phones with several sensors. The operating system supports features such as application installation, multimedia support and browsing.

Tablets are similar to smartphones but are physically larger. They are typically used when a larger display is needed or desired and they may also support longer battery life.

Companion devices and some IoT appliances are computer-powered devices commonly used together with a smartphone or tablet to extend the available functionality or to give access to the data on the phone or tablet in a more convenient way.

Wearables are devices that can be worn by consumers. These can act as a companion to existing devices or function independently. Watches and fitness bands are examples of popular wearables.

1.4 Types of Mobile Applications

There are three main types of mobile application:

- Native
- Browser-based
- Hybrid

Each type of application has certain advantages and disadvantages, requiring a business decision to be made before starting the application development.

Native applications are developed using platform specific software development kits (SDKs), development tools and platform specific sensors and features. They are downloaded, installed and updated from supplier stores. These apps may need testing on all supported devices.

Native applications generally provide better performance, can fully utilize platform features and comply to the expectations for the platform they are developed for. The development cost is typically higher and additional challenges may apply such as the use of multiple platforms and the installation and testing on a large number of devices.

Browser-based applications are accessed through a mobile browser. Since these use the typical web-development technologies and browsers, multiple platform support is easy, and the development cost is usually lower.

There are four main ways in which mobile web applications are created:

- Mobile specific versions of websites and applications (these are also known as m(dot) sites). Usually this means that when a mobile browser addresses the application, a mobile version of

the application is provided. For example, facebook.com redirects to m.facebook.com when accessed from a mobile device.

- Responsive web apps ensure that the design adjusts to the form factor and screen size, usually expressed as view ports.
- Adaptive web apps adjust the design according to some predefined sizes. There are different designs for these sizes and the features available to the user are often adjustable.
- Progressive web apps allow shortcuts of specific web pages to be created on the mobile home screen. They appear like native apps and sometimes even can work offline.

Mobile web apps are created using common web technologies, which generally makes them easier to develop and manage compared to native and hybrid apps. They may however not be as feature-rich as native or hybrid apps and may have limited access to the platform's native Application Programming Interfaces (APIs). The access to mobile sensors is also limited. Installability testing on devices is not needed, but browser compatibility testing is required.

Hybrid applications are a combination of native app and web app. They use a native app wrapper which contains a web view to run a web application inside of a native app. These apps are downloaded from supplier stores and can access all of the device features. They are relatively easy to develop, update and maintain without updating the app installed on the device. The skills required for developing these apps are almost the same as for web development. Possible weak points for these apps include performance issues due to the use of a wrapper and possible divergences from the expected look and feel because of platform-specific aspects.

Native and hybrid apps are installed physically on a device and are therefore always available to the user, even when the device has no internet connection. In comparison, browser-based applications require internet access.

Some applications are pre-installed on the mobile device and others can be installed via various distribution channels, such as Apple App Store, Google Play Store, enterprise app stores (available only inside the enterprise network) and third-party app markets.

Testing of each of these application types may require a different approach. The parameters to consider include:

- Different types of devices to be supported
- Sensor and device features to be used
- Availability under various network conditions
- Installability, compatibility, performance efficiency, and usability

1.5 Mobile Application Architecture

There are multiple solutions to designing a mobile application.

Some of the considerations in choosing a particular architecture or design decision include:

- Target audience
- Type of application
- Support of various mobile and non-mobile platforms
- Connectivity needs
- Data storage needs
- Connections to other devices including IoT appliances

Architectural decisions include:

- Client-side architecture such as thin or fat client
- Server-side architecture such as single or multi-tier
- Connection type such as Wi-Fi, cellular data, Near Field Communication (NFC), Bluetooth
- Data synchronization methods such as store-and-forward, push and pull, synchronous and asynchronous communications

Thin client apps do not contain application code which is customized to the device and make minimal use of mobile operating system features. These apps typically use the web browser as the front-end and JavaScript as the language for implementing client-side logic.

Thick/fat client applications may have multiple layers of application code and may use mobile operating system features. These are typically Native or Hybrid applications.

The server-side architectures include the following possibilities:

- Single-tier architectures are monolithic and have all servers on the same machine. They are less scalable and harder to secure.
- Multi-tier architectures spread server-side components across various units. Two-tier architectures involve separate web and database servers, whereas three-tier architectures also include an application server. Multi-tier architectures allow separation of responsibilities, provide database specialization and provide better flexibility, scalability and security. However, they may be significantly more expensive to develop, manage and host compared to single-tier architectures.

There are various connection methods. A mobile device might be connected to the server via connection types such as Wi-Fi or via cellular data connections such as 2G, 3G, 4G, and 5G. Mobile applications typically operate in one of the following three modes:

- Never-connected apps work offline and don't need to be connected. A simple calculator is an example of such an app.
- Always-connected apps require a permanent network connection during operation. All mobile web applications fall into this category, although some can operate in a limited way when partially connected.
- Partially-connected apps require a connection for tasks such as data transfer but can operate for long periods of time without connection.

The synchronization of data between the client and the server can be conducted in the following modes:

- Continuous mode is where the data gets transferred as soon as it is submitted.
- Store-and-forward mode is where the data may be stored locally before being transferred, especially when no connectivity is available.

The data transfer can be performed in the following two approaches:

- Synchronous data transfer is performed when the calling function waits for the called function to complete before returning.
- Asynchronous data transfer is performed when the called server function returns immediately, processes the data in the background and calls back the calling client function once it completes the task. This give users more control. However, implementing

the handshake mechanism increases complexity concerning the availability of the client or the network when the server initiates the callback.

1.6 Test Strategy for Mobile Apps

Creating a test strategy for mobile devices requires the tester to take into account all the parameters listed so far in this chapter. In addition, the risks discussed in this section and the challenges described in section 1.7 must also be considered.

Typical risks are, for example:

- Without knowing the device proliferation data in a particular geographic location, one cannot choose the devices on which the app needs to be tested in a sustainable fashion.
- Without knowing the type of business model, one cannot test whether the application behavior is a good fit for that business model.

Creating a test strategy for mobile application testing additionally needs to consider the following specific risks and challenges:

- The variety of mobile devices with device-specific defects on some of them.
- The availability of devices in-house or via the use of external test labs.
- The introduction of new technologies, devices and/or platforms during the application life cycle.
- The installation and upgrade of the app itself via various channels, including preserving app data and preferences.
- Platform issues which might impact the application.
- Network coverage and its impact on the app in a global context.
- The ability to test using the networks of various service providers.
- The use of mobile emulators, simulators and/or real devices for specific test levels and types of test.

These challenges are described in greater detail in the section 1.7.

The test strategy takes risks and challenges into account. For example:

- The test strategy may specify the use of mobile emulators/simulators in the early stages of development, followed by real devices in later stages. There are certain types of tests that can be performed on the mobile emulators/simulators but not all types of tests. More on this is described in section 4.3.
- The test strategy may consider the challenge posed by a large number of different devices by adopting one of the following approaches:
  - Single platform approach: Reduce scope to a single type of device, one OS version, one carrier and one network type.
  - Multi-platform approach: Reduce scope to a representative selection of devices and OS used by a majority of customers in the target market, based on mobile traffic or other analytical data.

- Maximum coverage approach: Cover all OS versions, devices, manufacturers, carriers and network types. This is basically exhaustive testing, which is usually not economically viable, especially when considering the multitude of devices and OS versions on the market.

- The test strategy may consider the challenge posed by the non-availability of devices, networks or real-life conditions by using external resources, such as:
  - Remote device access services. This is a way to access devices over the web which are not otherwise owned.
  - Crowd testing services. This is as a way to access a huge group of volunteers and their devices.
  - Personal networks such as friends and colleagues. This makes use of one's own private social network.
  - Bug hunting. This is gamified testing event using volunteers from the company or from the general public.

In addition to the test levels described in [ISTQB_CTFL_2018] the test strategy also considers the common types of testing used for mobile applications (see section 3.1) and any additional levels of testing required (see section 3.2).

1.7 Challenges of Mobile Application Testing

In the mobile world many additional challenges exist that are uncommon or uncritical in desktop or server software. Testers must be aware of these challenges and how they might impact the success of the application.

Typical challenges in the mobile world include:

- Multiple platforms and device fragmentation: Multiple OS types and versions, screen sizes and quality of display.
- Hardware differences in various devices: Various types of sensors and difficulty in simulating test conditions for constrained CPU and RAM resources.
- Variety of software development tools required by the platforms.
- Difference of user interface designs and user experience (UX) expectations from the platforms.
- Multiple network types and providers.
- Resource-starved devices.
- Various distribution channels for apps.
- Diverse users and user groups.
- Various app types with various connection methods.
- High feedback visibility resulting from bugs that have a high impact on users which may easily result in them publishing feedback on online market places.
- Market place publishing which requires additional approval cycles for publishing by market place owners such as Google Play or Apple App Store.

- Unavailability of newly launched devices requiring the use of mobile emulators/simulators

The impact of these challenges includes:

- Large numbers of combinations to be tested.
- Large numbers of devices required for testing, which drives up the cost.
- The need for backward compatibility to run the application on older versions of the platform.
- New features being released in every version of underlying operating system.
- Guidelines to be considered for the various platforms.
- Resource-starved CPUs as well as limited amount of memory and storage space.
- Varying bandwidth and jitter of various networks.
- Changes in the available upload and download speeds based on data plans.

The following two examples illustrate typical challenges and their potential impact:

- Different devices have different types of sensors and tests need to account for these. Every new sensor added to the hardware may require additional backward compatibility testing.
- Some of the network challenges can be dealt with appropriately, even under varying network conditions, by using appropriate caching and prefetching strategies. However, this comes at a cost; a large number of open connections can impact the server-side performance as most apps keep the user logged-in on the server.

1.8 Risks in Mobile Application Testing

The challenges mentioned in section 1.7 can appear in isolation or in combination with others. This may result in additional risks for a mobile application.

A tester must be able to contribute to the product risk analysis. Common risk analysis and mitigation methods, as discussed in [ISTQB_CTFL_2018], chapter 5.5, can also be applied in the mobile context. In addition, the following mobile-specific risks and mitigation strategies exist:

Risk	Possible mitigation
Market fragmentation	Choose an appropriate selection of devices for test execution, e.g., testing the most commonly used devices.
Cost of supporting multiple platforms	Perform analysis to understand most used platforms, thus avoiding testing of those no longer in scope.
Introduction of new technologies, platforms and devices	Use pre-production versions of those technologies.
Lack of availability of devices for test execution	Apply remote device access services or crowd testing services.
Risks from the expected usage patterns of mobile applications used while on the go	Applying appropriate testing approaches such as field testing

2. Mobile Application Test Types – 265 mins

Keywords

co-existence, compatibility, connectivity, cross-browser compatibility, interoperability, system under test (SUT), test type, usability

Learning Objectives for Mobile Applications Test Types

2.1 Testing for Compatibility with Device Hardware

MAT-2.1.1 (K2) Describe device-specific features and hardware which should be considered for testing.

HO-2.1.1 (H1) Test an app for several mobile device functionalities while the system under test (SUT) is in use to verify correct functioning of the SUT.

MAT-2.1.2 (K3) Prepare tests for the app's compatibility with screen sizes, aspect ratio, and screen density.

HO-2.1.2 (H3) Test an app on several mobile devices (virtual or physical) to show the impact of the resolution and screen size on the app's user interface.

MAT-2.1.3 (K2) Describe how tests can show the potential effects of device overheating on the system under test.

MAT-2.1.4 (K1) Recall different test types for testing of the various input sensors used in mobile devices.

MAT-2.1.5 (K1) Recall tests to be run for various input methods.

HO-2.1.5 (H0) Test an app for various types of inputs including keyboard-related tests with multiple installed keyboards, gesture-related tests and (optionally) camera-related tests.

MAT-2.1.6 (K2) Describe how tests can reveal user interface issues when changing screen orientation.

HO-2.1.6 (H3) Test an application to check the effect of orientation change on the functionality of the app, including data retention and correctness of the user interface.

MAT-2.1.7 (K3) Prepare tests for an app using typical mobile device interrupts.

HO-2.1.7 (H3) Test an app for several mobile device interrupts while the application is in use.

MAT-2.1.8 (K3) Prepare tests for changing the access permissions to the device features requested by the app.

HO-2.1.8 (H3) Test an app's permissions management by permitting and denying requested permissions and observing behavior when folders and sensor settings are denied at installation or changed after installation.

MAT-2.1.9 (K3) Prepare tests to verify the impact of an app on a device's power consumption and the impact of its power state on the app.

HO-2.1.9 (H3) Test an app under varying battery power levels to discover consumption data and establish performance under low and dead battery states.

2.2 Testing for App Interactions with Device Software

MAT-2.2.1 (K3) Prepare tests for the handling of notifications by the system under test.

HO-2.2.1 (H2) Test the effect of receiving notifications when an app is in the foreground and the background. Test the effect of changing notification settings on the app's functionality.

MAT-2.2.2 (K2) Describe how tests can verify correct functionality of quick-access links.

HO-2.2.2 (H3) Test an app for shortcut/quick-access functionality.

MAT-2.2.3 (K3) Prepare tests for the impact on an app of the user preference settings provided by an operating system.

HO-2.2.3 (H3) Test a running app by changing the input value options for the preferences provided by the operating system.

MAT-2.2.4 (K2) Distinguish between different tests required for native, web and hybrid applications.
HO-2.2.4 (H0) (Optional) Identify the tests needed for apps, depending on the app type.
MAT-2.2.5 (K1) Recall tests required for apps which are available on multiple platforms or operating system versions.
MAT-2.2.6 (K1) Recall tests required for co-existence and interoperability with other apps.

2.3 Testing for Various Connectivity Methods
MAT-2.3.1 (K2) Summarize the tests for connectivity testing, including those across networks, when using Bluetooth and when switching to flight mode.
HO-2.3.1 (H0) (Optional) Conduct tests on an application which is transferring data to the server when the phone switches between Wi-Fi and cell-data connectivity based on their available signal strengths.

2.1 Testing for Compatibility with Device Hardware

2.1.1 Testing for Device Features

Different types of devices with differing capabilities mean that compatibility testing has to be conducted on a large number of devices. This requires prioritization of target devices for testing. For prioritization market data, as discussed in section 1.1, is used to select a device portfolio most appropriate for the target market. The device portfolio selection usually is a compromise between market coverage, cost and risk.

Applications can be installed on different types of devices with the following features:

- Different methods for switching off
- Different ways to navigate
- Use of hard and soft keyboards
- Various hardware features such as:
  - Radio
  - USB
  - Bluetooth
  - Cameras
  - Speakers
  - Microphone
  - Headphone access

None of these features should interfere negatively with the application's operations.

Device features have many variations and can differ even between different device models made by the same manufacturer. They are commonly used to differentiate between market segments and can change quickly over time. For example, it is currently quite common for devices in the high-end and mid-range to have fingerprint sensors, while devices in the low-end do not. This changes over time. A few years ago, fingerprint sensors were not included in any mobile device at all. Due to this changeability, the tester needs a clear understanding of the devices and the features expected by its users. The tester needs to create the device portfolio and design corresponding tests accordingly.

Generally, it is not enough to test if the application works correctly with the expected features. In addition, it is required to test that the app still works as expected if a certain feature is absent. For

example, an app that supports the usage of front and rear camera should not crash if it is installed and executed on a device having multiple cameras, only one camera or no camera at all.

2.1.2 Testing for Different Displays

Device displays can have various screen sizes, viewport sizes, aspect ratios and resolutions measured in pixels per inch (ppi) and dots per inch (dpi). Device fragmentation requires prioritization to be performed. Tests should be created that exercise the user interface on various devices with different screen sizes, resolutions and aspect ratios most common in the target market.

Testing for different displays needs to be carried out to check the following:

- The app scales all user interface elements according to current screen density and size.
- User interface elements do not overlap.
- Usability or touch issues do not occur.
- There is no problematic shrinkage of images because of high dpi/ppi.

2.1.3 Testing for Device Temperature

Unlike desktop computers, mobile devices react differently to increases in device temperature.

Mobile devices could get overheated for a variety of reasons such as battery charging, intense workload, apps running in the background, continuous usage of cellular data, Wi-Fi or GPS.

Overheating can impact a device as it attempts to reduce heating and conserve battery levels. This may include a drop in the CPU frequency, the freeing up of memory, and the turning off parts of the system.

If this happens it can also impact the app functionality and therefore must be considered when planning testing. Tests must be designed to consume a lot of energy which leads to the generation of heat over a long uninterrupted period of time. The software under test must then show no unexpected behavior.

2.1.4 Testing for Device Input Sensors

Mobile devices receive a variety of inputs types from sensors which use, for example, GPS, accelerometers, gyroscopes, and 3-axis magnetometers or which react to pressure, temperature, humidity, heartbeat, light or touchless inputs.

Testing for different device input sensors checks the following:

- The app works as intended for each of the sensors available. For example, the app needs to be tested for various types of motion such as circular motion and back and forth motion (as in walking).
- Features that react to external lighting react correctly under various lighting conditions.
- Sound inputs and outputs respond correctly in conjunction with soft and hard volume buttons, microphones, wired and wireless speakers, and in various ambient sound conditions.
- Location position is accurate under the following conditions:
    - Switching GPS on and off.
    - Different GPS signal quality.
    - Where the app needs a fall back to various other methods of location determination, including Wi-Fi, cell tower location or manual location entry.

2.1.5 Testing Various Input Methods

Testing for different device input methods checks the following:

- Given that mobile phones allow for a variety of soft keyboards to be installed, the app is able to work with at least those provided by major device manufacturers and those which are widely used.
- The app ensures that the keyboard pops up by default with appropriate layout and keys when required.
- When a user places one or more fingers on the touch screen, the application interprets that pattern as a particular gesture or command. Typical gestures include press/touch, double touch, multi-touch, swipe, tap, double tap, drag, and pinch open/close.
- Each screen of the app needs responds correctly to the gestures or other means of input as appropriate for that screen and ignores all non-supported gestures or inputs.
- Cameras used by apps are able to capture images and videos, scan barcodes, QR codes and documents, and measure distances.
- Where front and back cameras are available, the appropriate camera is turned on by default. For example, where a video chat requires the front camera to be switched on by default, apps need to be tested in cases where the app uses the camera input and where it does not. Additionally, tests must ensure the software under test works correctly if only one (front or rear) camera is present instead of two. This is especially true if the software under test uses one particular camera and it is this one that is missing.

2.1.6 Testing for Screen Orientation Change

Motions sensors are used to detect changes in orientation and trigger a switch between landscape and portrait modes (and vice versa) with layout changes made in the user interface as necessary.

Tests after a change of screen orientation check the following:

- Correct usability and functional behavior when a switch to either portrait or landscape mode is performed.
- The app maintains its state.
- Input data fields retain already captured data.
- Output data fields display the same data while maintaining the current session.

Tests after a change of screen orientation change should not just focus on a single switch because rendering or state issues may not always show up after a single change. Tests should therefore be performed with several uninterrupted switches between portrait and landscape modes.

Tests that switches orientation several times in the various states of a user interface, with and without data, should be designed. The app should behave as expected, persisting the state without any loss or change of data.

2.1.7 Testing for Typical Interrupts

Common types of device interrupts include voice calls, messages, charger switched on, low memory and other notifications. User-initiated interrupts result from actions such as app switching or setting the device into standby while the app is running

Tests for interrupts check the following:

- The app handles the all of the interrupts mentioned above correctly without negative impact on app behavior.
- The app continues to function correctly, preserving its state, data and sessions regardless of which interrupt occurs.
- Where the device has a blocking “do-not-disturb” mode which suppresses notifications, the app must ensure that the various conditions are used correctly. These tests must also be performed when the “do-not-disturb” mode is turned off after having been active for a long period of time. This results in many notifications being received at once.
- Test should be designed for receiving interrupts during app usage to make sure that the interrupts do not have a negative impact. For example, answering a phone call while using the app and the user being returned to the state where he was at the time of the interrupt.

2.1.8 Testing for Access Permissions to Device Features

Apps need access to various folders such as contacts and pictures and to sensors such as camera and microphone. When access is denied at installation or changed after installation it might impact the app behavior.

Tests for access permissions check the following:

- The app is able to work with reduced permissions; it asks the user to grant these permissions and does not fail in an unexplained manner.
- Permissions are only requested for the resources which are relevant to the app’s functionality; no broad permissions for unrelated resources are allowed.
- The app functionality responds correctly if a permission is withdrawn or rejected during installation.
- Any request for permission issued by the app is correct and justified.

To test for access permissions a tester needs to know why the app needs each permission and how functionality should be impacted if the permission is withdrawn or rejected during installation. Test should be designed for rejecting permissions during installation as well as granting permissions after installation.

2.1.9 Testing for Power Consumption and State

Tests for power consumption and state check the following:

- Battery power state and drainage-related defects.
- Data integrity under low power and dead battery conditions.
- Power consumption while the app is active and is under heavy and low use.
- Power consumption while the app is in the background.

These tests need to be planned carefully as these need to be run uninterrupted over an extended period of time. For example, the device may need to be left unattended with the app in the background or foreground but the device is not used. Tools such as log analyzers are needed to extract information about battery drain patterns.

2.2 Testing for App Interactions with Device Software

2.2.1 Testing for Notifications

There are various mechanisms used by the operating system to display notifications. Sometimes the operating system will either delay the display of the notifications or fail to display them at all in a bid to optimize power consumption. The following test conditions must be considered:

- The correct handling of notifications received when the app is in the foreground or background, especially under low battery conditions.
- If notifications allow direct interaction with the app content, (i.e., without opening the app itself), the user interaction must be provided by the app at a later time. If, for example, the user responds to a notification, then it must be possible to access that response from within the app at a later time.
- If notifications allow access to the app then the corresponding page of the app must be opened instead of the home screen when the notification contains a deep link to that page.

2.2.2 Testing for Quick-access Links

Quick-access links such as app shortcuts in Android and Force-touch or 3d-touch for iOS may be provided by the software under test. These features perform a subset of the application functionality from the home screen without actually launching the entire app.

The following test conditions must be considered:

- Where some of the features are only available on a particular version of the operating system, the system under test must behave correctly if it is installed on versions of the operating system which either offer or do not offer such features.
- The actions performed in quick-access links are reflected correctly in the app when opened.

2.2.3 Testing for User Preferences Provided by the Operating System

Any preferences (settings) provided to users by the operating system must be tested. It creates a negative experience for users if a certain preference setting is not respected by the app. For example, if the device is set to mute, the app should not play sounds.

The following test conditions must be considered:

- Users can amend typical preference options such as sound, brightness, network, power save mode, date and time, time zone, languages, access type and notifications.
- The apps adhere to the set preferences by behaving accordingly.

2.2.4 Testing for Different Types of Apps

Specific tests may be performed depending on type of mobile app (see section 1.4). The following test conditions must be considered:

- For native apps:
  - Device compatibility
  - Utilization of device features
- For hybrid apps:
  - Interaction of the app with the device native features
  - Potential performance issues due to the abstraction layer
  - Usability (look and feel) compared to native apps on the platform in question
- For web apps:
  - Testing to determine cross-browser compatibility of the app to various common mobile browsers
  - Functionality is not impacted due to various JavaScript engines
  - Utilization of OS features (e.g., date picker and opening appropriate keyboard)
  - Usability (look and feel) compared to native apps on the platform in question

2.2.5 Testing for Interoperability with Multiple Platforms and Operating System Versions

Software companies often support apps on multiple operating systems. Each mobile operating system has its own limitations which need to be taken into account when testing apps. Testers must be aware of the specifics of each platform tested to ensure the app works as intended whilst still conforming to the look and feel of the platform.

The following test conditions must be considered:

- Handling of interrupts, notifications and optimizations (e.g., for energy saving).
- Correct functionality where multi-platform applications share some code or have been created using cross-platform development frameworks. Note that if the applications do not share code, then it is like testing two different applications and everything needs to be tested.
- Testing for backward compatibility if a platform uses different operating system versions.
- Testing new or changed features made to platforms. For example, in Android the introduction of the Doze framework required testing on the various versions of the operating system which support this framework and those that don't.

2.2.6 Testing for Interoperability and Co-existence with other Apps on the Device

It is quite common for apps to interact with each other when installed on a device. Typical examples are the contact and email apps.

The following test conditions must be considered:

- Data transfer between the system under test and the utilized app is correct.
- There is no harm done to any user data stored within a utilized app.
- Conflicting behaviors. For example, an app might turn off GPS to save energy, while another app turns on GPS automatically.

With millions of apps in the market, co-existence cannot realistically be tested for all of them. Nevertheless, such potential issues should be considered and tested according to their risk.

2.3 Testing for Various Connectivity Methods

Mobile devices can use various methods to connect to networks (see section 1.5). These include cellular networks such as 2G, 3G, 4G and 5G, as well as Wi-Fi and other wireless connection types such as NFC or Bluetooth.

The following alternatives should be considered when performing tests for connectivity:

- Device emulators/simulators can simulate various network connections and some remote device access service providers include them within their features. Emulators/simulators are, however, of limited value.
- Setting up your own mobile network to support various connection types and then applying bandwidth manipulation. This is a very costly alternative.
- Field testing is potentially more cost-effective alternative, but is limited with regard to the reproduction of tests.

In real-world usage, connectivity methods differ. Users can be continuously connected using one particular mode, or they can switch between modes, such as from Wi-Fi to cellular (e.g., when a user leaves home while using the app). The user can switch between various Wi-Fi/cellular networks and versions, as well as between GSM cells. While on the move they may even hit dead spots with no network at all. Furthermore, the user can deliberately disconnect by, for example, switching to flight mode.

Connectivity testing must ensure that the following test conditions are considered:

- Correct app functionality with different connectivity modes.
- Switching between modes does not cause any unexpected behavior or data loss.
- Clear information is given to the user if functionality is restricted due to limited or no network connection or if bandwidth is low. The message should state the limitations and their reasons.

3. Common Test Types and Test Process for Mobile Applications – 200 mins

Keywords

abnormal end, accessibility, code injection, exploratory testing, field testing, heuristic, installability, performance efficiency, performance testing, post-release testing, security testing, session-based test management, stress testing, test level, test process, test pyramid, tour, usability lab, usability testing

Learning Objectives for Common Test Types and Test Process for Mobile Applications

3.1 Common Test Types applicable for Mobile Applications

MAT-3.1.1 (K3) Prepare installability tests for mobile apps.
MAT-3.1.2 (K3) Prepare stress tests for mobile apps.
MAT-3.1.3 (K2) Give examples of security issues related to mobile apps.
MAT-3.1.4 (K1) Recall time and resource behavior considerations for mobile apps.
MAT-3.1.5 (K3) Prepare usability tests for mobile apps.
HO-3.1.5 (H2) Choose a tour, a mnemonic or a heuristic for usability testing an app using session-based test management.
Note: HO-3.1.5 and HO-3.3.1, HO-3.3.2 and HO-3.3.3 may be combined.
MAT-3.1.6 (K1) Recognize the type of tests required for database testing of mobile apps.
MAT-3.1.7 (K2) Summarize the tests required for internationalization (globalization) and localization testing of mobile apps.
MAT-3.1.8 (K2) Summarize the need for accessibility testing in mobile application testing.

3.2 Additional Test Levels applicable for Mobile Applications

MAT-3.2.1 (K2) Describe the additional test levels, such as field testing, and the associated extra activities required for effective mobile application testing.
MAT-3.2.2 (K2) Describe the tests required for carrying out application store approval for publishing apps.

3.3 Experience-based Testing Techniques

MAT-3.3.1 (K1) Recall session-based test management, personas, and mnemonics in the context of exploratory mobile testing.
HO-3.3.1 (H2) Choose a mnemonic (or part thereof) which is specific to mobile application testing for testing of an app using session-based test management.
Note: HO-3.1.5 and HO-3.3.1, HO-3.3.2 and HO-3.3.3 can be performed together.
MAT-3.3.2 (K2) Describe the usage of tours and heuristics as exploratory techniques for mobile application testing.
HO-3.3.2 (H2) Choose a mobile specific heuristic to test mobile application.
Note: HO-3.1.5 and HO-3.3.1, HO-3.3.2 and HO-3.3.3 can be combined.
MAT-3.3.3 (K3) Make use of a mobile specific tour (such as the Feature tour) to test a mobile app.
HO-3.3.3 (H2) Choose a mobile specific tour to test a mobile application.
Note: HO-3.1.5 and HO-3.3.1, HO-3.3.2 and HO-3.3.3 can be combined.

3.4 Mobile Test Process and Approaches

MAT-3.4.1 (K2) Match the test process, as described in [ISTQB_CTFL_2018], to the needs of mobile application testing.
MAT-3.4.2 (K2) Describe the approaches to testing at each test level, specific to mobile application testing.

3.1 Common Test Types Applicable for Mobile Application

3.1.1 Installability Testing

Testers need to focus on installation, update and de-installation of the app using the following approaches:

- Application_stores

  The installation process may be different depending on the users of the app. The users could install the app from market place stores such as the Google Play Store or Apple's App Store. The users of enterprise apps will be required to perform installation tests via a link, or a distribution service such as HockeyApp or App Center.

- Sideloading (copying and installing app)

  Some operating systems provide the option of installing the application by copying it to a mobile device and installing it from the file.

- Desktop applications

  Desktop applications such as Apple iTunes (for iOS) or Android App Installer are available for installing apps on the smartphone. The tester needs to download the app in this application and use a cable to install it from there to the smartphone. Most of these desktop applications also allow de-installation of the app.

Installation can be performed using the following methods:

- OTA (Over-the-Air) via Wi-Fi or Cellular Data
- Data cable

Some of the test conditions that can be considered include:

- Installation, de-installation and upgrade on internal and external memory (if supported).
- Re-installation of app when the "retain app data" option was chosen during the previous de-installation.
- Re-installation of app when the "retain app data" option was not chosen during the previous de-installation.
- Cancelling or interrupting the installation or de-installation, for example, by shutting down the mobile device during the process or disconnecting from the internet.
- Resuming interrupted installation, de-installation and upgrade after cancelling or interrupting.
- Permissions-related testing. For example, some apps request permission to use the address book. This important test must verify app behavior if the user denies permission. For example, is there a corresponding message sent to the user?
- Update the app and verify that no data is lost.

Some apps require jailbroken (iOS) or rooted (Android) devices which give the user the administrative rights over the device. Most platform providers do not support jailbreaking/rooting as it may have legal consequences. An app not requiring jailbreaking/rooting may not need to be tested for the jailbreaking/rooting devices.

3.1.2 Stress Testing

Stress testing is focused on determining the performance efficiency of the application when subjected to conditions beyond normal load. The stress test is this context is targeted only at the mobile device. Stress tests of the backend are described in the ISTQB® Performance Testing syllabus ([ISTQB_CTFL_PT_2018]) so that further information can be referenced there as needed.

Some of the test conditions that can be considered for stress testing include:

- high CPU-usage
- out-of-memory
- low disk space
- battery stress
- failures
- poor bandwidth
- very high number of user interactions (real world network conditions may need to be simulated for this)

Some of these stressful conditions can be created using tools such as Monkey. This is a command line tool that runs over the ADB shell command line [URL3] or, if possible, manually, e.g., by using big files or other apps with high CPU-usage or memory consumption.

3.1.3 Security Testing

Since security testing is a complex topic, the ISTQB® has a separate specialist syllabus on this subject [ISTQB_CTAL_SEC_2016]. Principal security issues for mobile apps include:

- Access to sensitive data on the device.
- Unencrypted information transfer or unsafe storage.

Some of the test conditions that can be considered for security testing include:

- Testing inputs for code injection and overflow.
- Encryption of transferred data.
- Encryption of locally stored data.
- Deletion of temporary data after use or after an abnormal end.
- Clearing text in password fields.

Top 10 mobile related vulnerabilities from the Open Web Application Security Project (OWASP) should also be explored [URL2].

3.1.4 Performance Testing

If the user installs the app and it does not appear fast enough (e.g., less than or equal to 3 seconds) it may get de-installed in favor of another alternative app. Time and resource consumption aspects are important success factors for an app and performance testing is carried out to measure these aspects.

Performance efficiency needs to be tested on the device itself in addition to interaction with the backend system and other mobile devices.

Performance testing of the whole system should be performed as defined in the test strategy and is not mobile specific. Please refer to the ISTQB® specialist syllabus on performance testing [ISTQB_CTFL_PT_2018] for further details.

The performance test of the app itself should contain chronometry for the most important workflows. Some examples for the workflows of an online-banking app are: “Login”, “Change address” or “Bank transfer with PIN and TAN”. The tester should then compare this chronometry with similar apps.

Besides chronometric measures it is important to consider the perceived performance by the user. User experience can have a huge impact on how long the user is willing to wait for a certain function to complete.

3.1.5 Usability Testing

Usability is very important for mobile apps because data shows that a large number of users de-install their apps within a few minutes of installing because of poor usability or performance, see [URL4].

Due to this it is recommended that user experience (UX) design considers the look and feel of the platform which the app is to be used on. If the UX does not conform with the user's expectations for their platform of choice, it can have a strong negative impact. Thus, a tester should be aware of the look and feel on the platform used.

Usability tests can be conducted by a tester using various available heuristics and test tours. Considering personas is also a helpful support for usability testing. If required, a usability lab can also be used for this purpose.

In projects, findings identified during the usability test are mostly just findings and not defects. The tester must have the ability to explain the findings to the team, Product Owner or similar stakeholders. To achieve satisfactory usability, an app should:

- be self-explanatory and intuitive.
- allow user mistakes.
- be consistent in wording and behavior.
- abide by the design guidelines of the platforms.
- make needed information visible and reachable in each screen size and type.

Please refer to the ISTQB® specialist syllabus in usability testing [ISTQB_FLUT_2018] for further details.

3.1.6 Database Testing

Many apps need to store data locally using various data storage mechanisms such as flat files or databases. Some of the test conditions to be considered for the database testing of mobile apps include:

- Validation of data storage issues:
  - Synchronization
  - Upload conflicts
  - Data security
  - Constraints on the data
  - CRUD (Create/Read/Update/Delete) functionality
  - Search
- Data integration testing for data provided by the device (e.g., contacts) or by third-party apps (e.g., pictures, videos and messages).
- Performance of storing the data on the device.

3.1.7 Globalization and Localization Testing

Internationalization (I18N) /Globalization testing of the application includes testing an app for different locations, formats for dates, numbers and currency, and replacing actual strings with pseudo-strings.

Localization (L10N) testing includes the testing of an app with localized strings, images and workflows for a particular region. For example, Russian and German words could be much longer than those in other languages. Since mobile devices have different screen sizes and resolutions, limited screen sizes may lead to problems with translated strings. These issues should be checked as standard globalization/localization tests.

A very important aspect to be checked is the date format used, such as YEAR – MONTH – DAY or DAY – MONTH – YEAR.

3.1.8 Accessibility Testing

Accessibility testing is performed to determine the ease by which users with disabilities can use a component or system. For mobile apps this can be done using device accessibility settings and testing the app for each setting.

Accessibility guidelines are available from platform vendors and these should be used. For example, both Google [URL5] and Apple [URL6] have published accessibility guidelines for their respective platforms. Taking feedback from people who require accessibility is also helpful.

For mobile web an accessibility guide has been published by the W3C, which should be considered [URL7].

3.2 Additional Test Levels applicable for Mobile Applications

In addition to the usual levels of testing from component through to acceptance testing described in [ISTQB_CTFL_2018], there is also a need for additional test levels for mobile application testing.

3.2.1 Field Testing

Some mobile applications need field testing to ensure that they function correctly in the expected usage scenario of real users. This could include testing on various networks and on different types of communication technologies such as Wi-Fi or cellular data.

Field tests should include the use of mobile towers, networks, Wi-Fi, and cellular data switching while the app is in use. Tests should be performed with varying download speeds and signal strengths, and include the handling of blind spots.

Field testing requires careful planning and the identification of all items required to perform the tests, such as appropriate device types, Wi-Fi, cellular data plans on various carriers and access to various modes of transport required to give adequate coverage. In addition, the routes and modes of transport, and the time of the day when the tests are to be executed need to be scheduled.

Usability of an app is another important aspect that needs to be covered while conducting field testing. Tests should incorporate environmental factors such as temperature and similar conditions related to usage scenario.

3.2.2 Testing for Application Store Approval and Post-release Testing

Before an app is sent for publishing some checklist/based tests must be passed to assure the approval of the application stores. If the release is an upgrade, then upgrade related tests should also be run.

Checklists are typically based on guidelines, such as those specific to operating systems, for user interface design, and for using the libraries and APIs provided by application stores.

The approval process may take some time after submission. If any issues are found during the approval process, a new version may need to be submitted, which will require additional time to resolve. This situation requires careful consideration during project planning and testing.

A further level of testing is “post-release” testing. Testing at this level includes downloading and installing the application from application stores.

3.3 Experience-based Testing Techniques

3.3.1 Personas and Mnemonics

Personas are fictional characters which represent real customers. They have motivations, expectations, problems, habits and goals and it helps to use them when real user behavior needs to be mimicked.

A persona could have a name, gender, age, income, an educational background, and a location. In a mobile context they may use other apps, check their mobile device x times an hour and can have other devices and personal traits.

A mnemonic is a memory aid to remember something. In the context of testing, every letter in a mnemonic stands for a technique, a testing method or a focal point for testing. An example of a mnemonic is SFiDPOT [URL8]. Letters in the mnemonic have the following meanings:

S – Structure (e.g., user interface elements, other application elements and their order and call hierarchy)

F – Function (e.g., desired features are working, available, and functioning according to the requirements etc.)

i - Input (e.g., all required inputs are available and processed as they should be, such as inputs from the keyboard, sensors, and camera)

D – Data (e.g., the data is stored (also on SD card), modified, added, and deleted as defined in the requirements)

P – Platform (e.g., the specific operating system functions are available depending on device settings, includes store for downloading the app)

O – Operations (e.g., the activities of the normal user are available, such as moving between mobile carrier networks and Wi-Fi)

T – Time (e.g., handling and display of time zones, time, and dates)

A mnemonic and heuristic specifically dealing with mobile is I SLICED UP FUN [URL9]. Letters in the mnemonic have the following meanings

I – Inputs

S – Store

L – Location

I – Interactions and interruptions

C – Communication

E – Ergonomics

D – Data

U – Usability

P – Platform

F – Function

U – User scenarios

N – Network

3.3.2 Heuristics

A heuristic approach is a “rule of thumb” approach to problem solving, learning and discovery that employs a practical method. This does not guarantee to be optimal or perfect, but can be considered as sufficient for achieving the immediate goals.

There are many heuristics for mobile testing. Most mnemonics can be used as heuristics, but not every heuristic is a mnemonic.

3.3.3 Tours

Tours are used in exploratory testing to enable an application to be explored from a specific point of view and focus. They can be performed to understand how an application works and to create models for the workflow. Tours provide an effective method for field testing.

An example of a tour is the Landmark tour where a user mimics the visits of a tourist in a city by going to well-known landmarks. The following table shows how visits made on the tour can be used as analogies for the steps to follow in mobile testing.

Visits in the Landmark tour	Analogy for mobile testing
The historical quarter	Legacy code
The business district Rush hour	Business logic of the app App startup and shutdown
The tourist quarter	Part of the app used by newcomers
The hotel quarter	Parts of the app which are only active in sleep mode

Combining session-based testing (see section 3.3.4) with tours, including the use of heuristics and mnemonics, helps in enhancing the effectiveness of mobile application testing.

The following table shows some good examples of tours for app testing and the areas they cover for giving testing ideas. Some of these are to be found in [Kohl17].

Tour for app testing	Subject covered
Supermodel	Look and feel and usability
Landmark	The most important features in the app
Sabotage	Robustness
Feature	New features
Scenario	Whole workflow in the app in combination with user stories
Connectivity	Connectivity used, such as Wi-Fi, GSM
Location	Correct language, dates, numbers
Light	Visibility in different lighting conditions, such as dark, outside, red light.
Low battery	Data losses in the app caused by of low energy levels.
Further tours for app testing	**Subject covered in [Kohl17]**
Gesture	Use all gestures wherever possible
Orientation	Change orientation
Change your mind	Go back
Motion	Do various types of movements
Location	Move around
Connectivity	Change connection types or locations by moving
Comparison	Compare with other type of devices
Consistency	Check consistency of screens, GUI

3.3.4 Session-Based Test Management (SBTM)

Session-based test management (SBTM) enables exploratory testing to be managed in a time-boxed fashion. A session consists of three tasks:

- Session setup
- Test design and execution
- Issue investigation and reporting

SBTM typically uses a session sheet containing a test charter that provides test objectives. Additionally, the session sheet is used to document the test execution activities carried out.

Exploratory testing is an experience-based test technique which can be an effective approach for testing mobile applications. Experience-based test techniques are described in [ISTQB_CTFL_2018].

3.4 Mobile Test Process and Approaches

3.4.1 Test Process

The main activities of the ISTQB® test process are described in [ISTQB_CTFL_2018] and are also applicable to mobile application testing.

There are additional aspects which are specific to mobile testing which should always be considered as part of the ISTQB® test process.

Main group of activities in the test process	Typical areas to consider for mobile testing	Syllabus reference
Test planning	• Device combinations that need to be tested. • Use of mobile emulators and mobile simulators as part of the test environment. • Special challenges in mobile app testing. • Test types specifically required for mobile application testing.	• Section 4.3 • Section 1.7 • Section 3.2
Analysis and design	• App Stores Approval testing. • Field testing. • Device compatibility. • Kind of labs to be used. • Test types specifically required for mobile application testing.	• Section 3.2.2 • Section 3.2.1 • Section 3.2
Test Implementation and Test Execution	• Field testing. • Download and installability post-release testing. • Experience-based techniques.	• Section 3.2.1 • Section 3.1.1 [ISTQB_CTFL_2018]
Test Implementation and Test Execution	• Tests are based on platform-guidelines for the user interface and the application stores. • Tests based on guidelines will typically be run by the platform providers for their application store approval process. • It is recommended to run these as application provider before handing over to the platform providers in order to avoid possible rejection.	

3.4.2 Test Approaches

Mobile application testing includes activities to be performed by developers as well as testers.

Determining the appropriate depth of testing per test level, (i.e., component test, integration test, system test, field test, application store approval, post-release and user acceptance testing) are important for delivering good quality products. The depth of testing needed per test level depends on many factors, such as app architecture, app complexity and intended user audience.

Mobile development platforms provide a variety tools to support testing at the various levels. Understanding the tools and how they can be applied at a given level are very important. For example, a mobile simulator and/or mobile emulator can be used at the component testing level if there is a need to take advantage of the platform-provided framework and instrumentation APIs. In addition, mobile simulators and/or mobile emulators can be used at the system testing level when actual devices are not available. This enables testing for functionality, limited aspects of usability as well as the user interface.

Furthermore, early implementation can serve as a key point to ensure that the devices are setup correctly and all the prerequisites for the execution will be met on time.

Unit and integration tests are also important, as well as manual testing, (especially in the field-testing stage). It is very common for mobile apps to flip the Test Pyramid [Knott15]. This means that there can be many manual tests.

4. Mobile Application Platforms, Tools and Environment – 80 mins

Keywords

emulator, field testing, proximity-based testing, remote test lab, simulator

Learning Objectives for Mobile Applications Platforms, Tools and Environment

4.1 Development Platforms for Mobile Applications
MAT-4.1.1 (K1) Recall the development environments used for mobile application development.

4.2 Common Development Platform Tools
MAT-4.2.1 (K1) Recall some of the common tools supplied as part of application development platforms.
HO-4.2.1 (H1) Use tools from the software development kit to take screenshots, extract a log and simulate incoming events.

4.3 Emulators & Simulators
MAT-4.3.1 (K2) Understand the differences between emulators and simulators.
MAT-4.3.2 (K2) Describe the use of emulators and simulators for mobile application testing.
HO-4.3.2 (H1) Create and run a simulated/emulated device, install an app and execute some tests on it.

4.4 Setting up a Test Lab
MAT-4.4.1 (K2) Distinguish between various approaches to set up a test lab.

4.1 Development Platforms for Mobile Applications

Integrated development environments (IDEs) are available on the market for various mobile app developments. These IDEs have various tools that help in designing, coding, compiling, installing, de-installing, monitoring, emulating, logging and testing apps.

For example, Android Studio may be used for Android app development and for iOS app development Xcode may be used. These differ from normal IDEs by the added support they offer for the mobile platforms.

Some cross-platform development frameworks are also available which helps in the development of mobile applications. These run on multiple platforms and do not specifically require coding.

4.2 Common Development Platform Tools

Software development kits usually provide various utilities which are helpful in developing and testing applications. These utilities span a wide range of purposes, such as taking screenshots, extracting logs, sending random events and notifications to the device, monitoring various parameters such as memory and CPU utilization, and creating virtual devices.

Some examples of such tools are Android Virtual Device (AVD) Manager, Android Debug Bridge (ADB), and Android Device Monitor for Android and Instruments for iOS.

4.3 Emulators & Simulators

4.3.1 Overview of Emulators & Simulators

In the context of this syllabus, the terms emulator and simulator refer to mobile emulator or mobile simulator. The terms simulator and emulator are sometimes used interchangeably but incorrectly. For definitions please refer to the glossary in chapter 8.

A simulator models the runtime environment, whereas an emulator models the hardware and utilizes the same runtime environment as the physical hardware. Applications tested on a simulator are compiled into a dedicated version, which works in the simulator but not on a real device. Thus, it is independent of the real OS.

By contrast, applications compiled to be deployed and tested on an emulator are compiled into the actual byte-code that could be also used by the real device.

Simulators and emulators are very useful in the early stage of development as these typically integrate with development environments and allow quick deployment, testing, and monitoring of applications.

Simulators are sometimes also used as replacement for real devices in testing. However, this is even more limited than the usage of emulators, as the application tested on a simulator differs at byte-code level from the application that will be distributed.

Emulators are also used to reduce the cost of test environments by replacing real devices for some of the testing. An emulator cannot fully replace a device because the emulator may behave in a different manner than the mobile device it tries to mimic. In addition, some features may not be supported such as (multi)touch, accelerometer, and others. This is partly caused by limitations of the platform used to run the emulator.

4.3.2 Using Emulators and Simulators

Using emulators and simulators for mobile testing can be helpful for various reasons.

Each mobile operating system development environment typically comes with its own bundled emulator and simulator. Third party emulators and simulators are also available.

A tester may use whichever emulator or simulator suits their purpose. Using the emulator or simulator requires launching them, installing the necessary app on them and then testing the app as if it were on the actual device.

Usually emulators and simulators allow the setting of various usage parameters. These settings might include network emulation at different speeds, signal strengths and packet losses, changing the orientation, generating interrupts, and GPS location data. Some of these settings can be very useful because they can be difficult or costly to replicate with real devices, such as global GPS positions or signal strengths.

Connecting to the emulators for the purpose of installation might require using command line tools such as for Android Debug Bridge (ADB) for Android or connecting from within the integrated development environment as with Xcode or Android Studio.

4.4 Setting up a Test Lab

The following approaches are used to set up a mobile test lab:

On-premise lab

With an on-premise lab all the devices, emulators and simulators are located on site. Device selection can be done on the basis of various factors such as rank of the device (as found in Google or other analytics), operating system and versions, screen dimensions and density, availability and cost, special features, and importance in targeted audience.

On-premise lab advantages include the availability of devices for specific proximity-based testing and sensor specific aspects such as battery, touch, and enhanced security.

Setting up this type of lab may require large budgets depending on the devices to be procured and maintained. Additional challenges include timely availability and difficulties with testing in different locations and environments.

Remote test lab

These labs are important and helpful for testing when devices or networks are not physically available on site. Remote device access (RDA) allows access via a network connection to various devices hosted in the provider's data center. Each potential RDA provider needs to be evaluated for compliance with the requirements, especially for security.

Some remote labs provide the following additional features:

- Dedicated physical device versions (e.g., Samsung mobile devices lab).
- Generic devices for a particular operating system and version only.
- Robotic arms for performing touch and gesture related operations.
- Virtual private network (VPN) connections to give access to the device.
- Cellular connections with various cellular network providers.
- Automation tools and services.

Some of the factors to be kept in mind when using remote test labs include slow device responsiveness and limited options for interacting with devices such as multi-touch and gestures. This can be cost-effective for sporadic use, but is generally more expensive if used for extended periods of time for a wide range of devices.

Other factors include on-demand platform availability compared to the need to obtain access to missing devices in the local lab and scalability of the lab, as it can grow and shrink as the project evolves.

Test scenarios that include sensors such as NFC/Bluetooth or battery consumption are often hard to test in the cloud. However, the different geographical locations of remote labs may help with tests that need network and GPS connections.

A test lab can utilize either one or a combination of the two approaches, depending on the type of tests that need to be performed.

5. Automating the Test Execution – 55 mins

Keywords

device-based testing, test report, user-agent based testing

Learning Objectives for Automating the Test Execution

5.1 Automation Approaches
MAT-5.1.1 (K2) Distinguish between common automation approaches and frameworks for mobile application testing.

5.2 Automation Methods
MAT-5.2.1 (K2) Describe various automation methods for testing mobile apps.

5.3 Automation Tools Evaluation
MAT-5.3.1 (K1) Recall the various parameters to be considered during the evaluation of mobile testing automation tools.

5.4 Approaches for setting up an Automation Test Lab

MAT-5.4.1 (K2) Distinguish between common approaches of creating test labs with advantages and disadvantages with respect to test automation.

5.1 Automation Approaches

Various automation approaches and frameworks exist that can be used in mobile application testing. The choice of approach will partly be determined by the type of application.

Two common test automation approaches used are:

- User-agent based testing
- Device-based testing

User-agent based testing utilizes the user-agent identifier string sent by the browser to spoof a particular browser on a particular device. This approach can be used for executing mobile web applications. Device-based testing on the other hand involves running the application under test directly on the device. This approach can be used for all types of mobile applications.

The application type can also determine the test automation framework that would be suitable for that application. Mobile web can be tested using the usual web application automation tools on the desktop, whilst native apps might need specific tools. Platform providers may also provide automation tools dedicated for the platform.

Automation approaches used for conventional applications are often applicable to mobile applications as well. These include capture/playback, data-driven, keyword-driven and behavior-driven testing as described in the ISTQB® Foundation Level syllabus [ISTQB_CTFL_2018] and in the ISTQB® Advanced Level Specialist Test Automation Engineer syllabus [ISTQB_CTAL_TAE_2016].

Key capabilities that a mobile application testing framework should typically include are:

- Object identification
- Object operations

- Test reports
- Application programming interfaces and extendable capabilities
- Adequate documentation
- Integrations with other tools
- Independent of test development practice

5.2 Automation Methods

To develop automated tests, the tester needs to understand the automation script recording or creation mechanism, and how to access and interact with the application's graphical objects such as buttons, list boxes, and input fields.

Several methods exist for identifying a graphical object used for the mobile test automation. These include image recognition, OCR/text recognition, and object recognition (web or native, depending on the app type).

A Mobile Application Tester needs to not only practice the graphical object detection and identification, but also to understand which object identification method will be the most capable in enabling successful tests to be run on a large variety of mobile devices, in parallel and continuously.

Key differences between the script creation methods are:

Item of Comparison	Object Identification	Image/OCR Comparison
Reliability	As long as the identifier is constant the screen layout can be changed. The risk is that objects can be identified and interacted with in the code while being hidden from the user. This may lead to false negative test results.	Images can be scaled according to screen size, but tests will fail as soon as the layout changes.
User experience	Usually manual scripting is required, at least to improve recorded scripts for readability and maintainability.	Full GUI-based testing without the need for scripting.
Execution speed	Tends to be faster than Image/OCR comparison, especially when using native tools provided by the system manufacturer.	Tends to be slower due to the need to compare the screen pixel by pixel with a baseline image.
Maintenance	Depends on the quality of the test scripts.	Mainly in providing changed baseline images.
Authoring challenge	Knowledge required of the scripting language and of software design methods to build a sustainable automation solution.	Generation of baseline images, especially when app changes often.

5.3 Automation Tools Evaluation

To be successful in creating test automation solutions, the test automation teams need to choose an appropriate set of tools. Understanding the key differences of the available tools and their suitability for the project requirements needs to be considered (see also [ISTQB_CTAL_TAE_2016]).

The evaluation parameters for test automation tools can be broken into two categories:

- Organizational fit
- Technical fit

Organizational fit parameters are described in chapter 6.2 of the ISTQB® Foundation Level syllabus [ISTQB_CTFL_2018].

Technical fit parameters include the following:

- Test automation requirements and complexities such as the use of new features like FaceID, fingerprint and chatbots by the app.
- Test environment requirements, such as varying network conditions, import or creating test data, and server-side virtualization.
- Test reporting and feedback loop capabilities.
- The ability of the framework to manage and drive execution on a large scale either locally or in a test lab in the cloud.
- Integration of the test framework with other tools used in the organization.
- Support and documentation availability for current and future upgrades.

5.4 Approaches for setting up an Automation Test Lab

When performing mobile application testing, developers and testers have choices around the device test lab they would use to target their test automation against.

- On-premise device test lab
- Remote device test lab

Various combinations of these approaches can be applied. Their principal characteristics are described and compared in section 4.4.

On-premise device test labs are generally difficult and time consuming to maintain. Having devices locally in parallel with emulators and simulators would best serve the early development and testing phases of the mobile app.

When reaching a more advanced stage of the app development, teams need to perform full regression test, functional tests, and non-functional tests. These tests are best executed on a full device lab. This is where a remote device test lab is managed, continuously updated, and maintained in the cloud. Such remote device test labs complement an on-premise device test lab and ensure that sufficient combinations of device and operating system are available and up to date. By making use of commonly available remote device test labs, teams get access to a larger set of supported capabilities including richer test reports and advanced test automation capabilities.

Lastly, when executing at scale through a test automation framework or through a continuous integration job (CI), stability of the overall test lab is key for test efficiency and reliability. Such labs are typically designed to ensure that devices and operating systems are always available and stable.

Remote device test labs are not always necessary in the later development stages of the app. Well designed and maintained on-premise device test labs can be as good as or better than any remote device test lab.

6. References

6.1 ISTQB® Documents

- [ISTQB_CTFL_2018]:
ISTQB® Certified Tester – Foundation Level Syllabus – Version 2018
- [ISTQB_FLAT_2014]:
ISTQB® Certified Tester – Foundation Level Extension Syllabus – Agile Tester – Version 2014
- [ISTQB_FLUT_2018]:
ISTQB® Certified Tester – Foundation Level Specialist Syllabus – Usability Testing – Version 2018
- [ISTQB_CTFL_PT_2018]:
ISTQB® Certified Tester – Foundation Level Specialist Syllabus – Performance Testing – Version 2018
- [ISTQB_CTAL_SEC_2016]:
ISTQB® Certified Tester – Advanced Level Specialist Syllabus – Security Testing – Version 2016
- [ISTQB_CTAL_TAE_2016]: ISTQB® Certified Tester – Advanced Level Specialist Syllabus - Test Automation Engineer - Version 2016
- [ISTQB_GLOSSARY]:
ISTQB®'s Glossary of Terms used in Software Testing, Version 3.2

6.2 Referenced Books

- [Knott15] Knott, D., "Hands-On Mobile App Testing", Addison-Wesley Professional, 2015, ISBN 978-3-86490-379-3
- [Kohl17] Kohl, J. , "Tap into mobile application testing", leanpub.com, 2017, ISBN 978-0-9959823-2-1

6.3 Further Books and Articles

- Boris Beizer, "Black-box Testing", John Wiley & Sons, 1995, ISBN 0-471-12094-4
- Rex Black, "Agile Testing Foundations", BCS Learning & Development Ltd: Swindon UK, 2017, ISBN 978-1-78017-33-68
- Rex Black, "Managing the Testing Process"(3e), John Wiley & Sons: New York NY, 2009, ISBN 978-0-470-40415-7
- Hans Buwalda, "Integrated Test Design and Automation", Addison-Wesley Longman, 2001, ISBN 0-201-73725-6
- Lee Copeland, "A Practitioner's Guide to Software Test Design", Artech House, 2003, ISBN 1-58053-791-X
- Rick David Craig, Stefan P. Jaskiel, "Systematic Software Testing", Artech House, 2002, ISBN 1-580-53508-9

6.4 Links (Web/Internet)

Disclaimer: All Links working as of 5 January 2019

- [URL1] http://gs.statcounter.com/
- [URL2] www.owasp.org
- [URL3] https://developer.android.com/studio/test/monkey
- [URL4] https://www.google.de/amp/s/techcrunch.com/2016/05/31/nearly-1-in-4-people-abandon-mobile-apps-after-only-one-use/amp/
- [URL5] https://www.google.com/accessibility/
- [URL6] https://www.apple.com/uk/accessibility/
- [URL7] https://www.w3.org/WAI/standards-guidelines/mobile/
- [URL8] https://www.slideshare.net/karennjohnson/kn-johnson-2012-heuristics-mnemonics
- [URL9] http://www.kohl.ca/articles/ISLICEDUPFUN.pdf

7. Appendix A – Learning Objectives/Cognitive Level of Knowledge

The following learning objectives are defined as applying to this syllabus. Each topic in the syllabus will be examined according to the learning objective for it.

7.1 Level 1: Remember (K1)

The candidate will recognize, remember and recall a term or concept.

Keywords: Identify, remember, retrieve, recall, recognize, know

Examples:

Can recognize the definition of “failure” as:

- “Non-delivery of service to an end user or any other stakeholder” or
- “Deviation of the component or system from its expected delivery, service or result”

7.2 Level 2: Understand (K2)

The candidate can select the reasons or explanations for statements related to the topic, and can summarize, compare, classify, categorize and give examples for the testing concept.

Keywords: Summarize, generalize, abstract, classify, compare, map, contrast, exemplify, interpret, translate, represent, infer, conclude, categorize, construct models

Examples:

Can explain the reason why test analysis and design should occur as early as possible:

- To find defects when they are cheaper to remove
- To find the most important defects first

Can explain the similarities and differences between integration and system testing:

- Similarities: the test objects for both integration testing and system testing include more than one component, and both integration testing and system testing can include non-functional test types
- Differences: integration testing concentrates on interfaces and interactions, and system testing concentrates on whole-system aspects, such as end-to-end processing

7.3 Level 3: Apply (K3)

The candidate can select the correct application of a concept or technique and apply it to a given context.

Keywords: Implement, execute, use, follow a procedure, apply a procedure

Examples:

- Can identify boundary values for valid and invalid partitions
- Can select test cases from a given state transition diagram in order to cover all transitions

Reference (for the cognitive levels of learning objectives):

Anderson, L. W. and Krathwohl, D. R. (eds) (2001) A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives, Allyn& Bacon: Boston MA

8. Appendix B – Glossary of Domain-Specific Terms

Glossary Term	Definition
2G	2nd generation of mobile wireless telecommunication technology.
3d-touch	See “Force Touch”
3G	3rd generation of mobile wireless telecommunication technology.
4G	4th generation of mobile wireless telecommunication technology.
5G	5th generation of mobile wireless telecommunication technology.
ADB	Android Debug Bridge (ADB) - command-line tool that allows communication with a device.
advertisement-based apps	An app monetization model where the development organizations earn money by advertisements shown within the app.
Android Device Monitor (ADM)	A standalone tool that provides a user interface for Android app debugging and analysis tools.
Android Studio	The official integrated developer environment (IDE) for Android. Android Studio provides tools for building apps on every type of Android device.
app shortcut	A shortcut to a specific set of actions defined in an application by application developers on Android 7.1 or higher.
application store	An application distribution platform where developers can upload their applications and users can search for applications to download and install them on their platform.
aspect ratio	The ratio of width to height of a display or image.
asynchronous communication	A type of communication in which data can be transmitted intermittently rather than in a steady stream.
AVD	Acronym for Android Virtual Device.
backend system	A server system that provides functionality for other systems.
background app	An app that is running in the background.
backward compatibility	The capability of an app to work on previous versions of platforms.
barcode	An optical, machine-readable representation of data.
basic phone	A mobile phone with minimal feature such as making calls, storing phone numbers, sending SMS, clock, and alarm.
blind spot	A location without wireless telecommunication network.
blocking/do-not-disturb mode	An operational mode of mobile devices that can be activated by the user to suppress certain features - common notifications and voice calls.
Bluetooth	A near-range wireless communication technology.
byte-code	An instruction set designed for efficient execution by a software interpreter. Also called portable code or p-code.
cellular data	Data transferred over a cellular network.
cellular network	A cellular network is a network created of multiple independent but connected cells.
companion device	A computer device designed to work in cooperation with a dependent smart device.
CPU frequency	The processor clock rate.
cross-platform development framework	A framework to develop an app for various platforms using the same code base.
CRUD	Mnemonic for Create/Read/Update/Delete which is applied to data.
data integrity	The accuracy and consistency of data over its entire life-cycle, including storage, processing, and retrieval.

Glossary Term	Definition
data synchronization	The process of bringing data into the same state across two or more sources.
data validation	The evaluation if the data is correct, accurate, consistent and useful.
dead spot	See “blind spot".
device fragmentation	The diversity of available device, their unique hardware configuration, and the impact thereof on the apps and user experience.
DPI/PPI	Acronym for Dots/pixels per inch - a number expressing the density of a display, either in dots or pixels.
emulator	A software application that mimics the behavior of hardware.
enterprise app	An application created to be used internally within an organization and not intended for public use.
external memory	An additional memory that is added to the device via a standard interface. Currently SD-Cards are most common for mobile phones.
fat client	In client/server applications, a client which has been designed to handle some or most of the data processing.
feature phone	A class of mobile phones which provides more functions than a basic phone, e.g., a browser, but does not provide the full functionality of a smartphone.
flat file	A file having no internal hierarchy.
flight mode	A special operation mode for mobile devices where the radio transmitters are deactivated to prevent interference with flight operation/communication systems.
Force touch	A technology developed by Apple Inc. that enables trackpads and touchscreens to distinguish between different amounts of force being applied to their surfaces.
foreground app	An app that is run in the foreground of the device for direct user interaction.
freemium app	A business model in which users pay nothing to download the app and are offered optional in-app purchases.
gesture	A certain interaction pattern, such as a pinch or swipe, to activate defined functions of the device. For example, a pinch is commonly used to zoom in an out on the smart device screen.
globalization	See internationalization.
GPS	Acronym for Global Positioning System - around the globe a network of satellites sends out time signals. By including the signal of at least 3 satellites a receiver can calculate its relative position to the satellites via triangulation.
GSM	Acronym for GSM (Global System for Mobile Communications, originally Groupe Spécial Mobile) is a standard developed by the European Telecommunications Standards Institute (ETSI) to describe the protocols for second-generation digital cellular networks used by mobile devices. Currently the most common standard for mobile communication in the world.
GSM cell	A part of a GSM network that can be identified by its unique cell ID
hybrid app	An application combining native and web technologies. Usually a hybrid app uses a native frame to be installed on the device to interact with device libraries and alike. Additionally, content is shown which is received from a web server.
I18N	Numeronym (number-based word) for Internationalization.

Glossary Term	Definition
IDE	Integrated development environment - A software application that provides comprehensive facilities to computer programmers for software development.
in-app purchase	Extra content and features available directly from an app.
instruments	A performance analysis and testing tool included as part of the Xcode tool set.
internal memory	A memory that is included in the device hardware.
internationalization	The process of preparing an application to accommodate for various localized versions.
interrupt	An event which occurs during another event.
IoT appliance	Internet of Things. A device or, for example, a sensor of interest connected to the internet.
jailbreaking	A privilege escalation for the purpose of removing software restrictions imposed by an operating system. Term usually used on iOS. Similar to rooting an Android.
L10N	Numeronym for localization.
landscape mode	The device orientation in which the display width is larger than the height.
library	A collection of non-volatile resources used by computer programs i.e., functions of the application.
localization	The process of adjusting an app or product to a certain region by actions such as translation and format-adjustments.
look and feel	Visual and emotional impression of something.
Mnemonic	A memory aid.
Mobile Application Mobile Testing	Testing mobile apps.
mobile device type	A classification of mobile devices by their basic features. Common classes include basic phone, feature phone, smartphone, phablet, tablet, and wearable.
mobile emulator	Virtual representation of a hardware platform. For example, the Android emulator is virtual hardware that runs a real android OS image. The very same OS image could be deployed to hardware and will work, as it is the real OS.
mobile OS	An operating system especially designed for mobile devices.
mobile platform	An ecosystem around a mobile operating system, usually including development tools, the operating system itself and an application distribution channel.
mobile space	An encapsulating term that includes anything in regard to mobile device technology, from the market and its players to the devices and apps.
mobile simulator	A virtual runtime environment. For example, the iOS simulator pretends to be iOS but actually is not a real iOS.
multi-platform applications	Applications designed and developed to run on multiple platforms using the same code base for all platforms.
multi-tier	A backend application design approach where 2 or more servers provide specialized functionalities.
multi-touch	A type of interaction with a device using various touch events in parallel.
native app	An application especially developed for a certain platform, usually using platform APIs and platform-provided development tools.

Glossary Term	Definition
NFC	Near Field Communication - a close range radio communication technology.
notification	An announcement sent out by the device.
OCR	Optical Character Recognition. The recognition of images of text contained in an electronic picture and its conversion to machine-encoded text.
on-premise lab	A lab that is physically located at the same place as the user of the lab.
orientation	The placement of an object in the mobile commonly used to express the way the device is used. It can either be landscape or portrait.
OTA	Over the air. Data transmission via radio signals, commonly used to refer to app installation to a device directly from a source not connected via cable.
overflow	A situation where the incoming data exceeds what can be accommodated.
paid app	An app that is monetized by selling it in app stores.
Persona	A model/archetype for a certain user group.
portrait mode	A device orientation in which the display height is larger than the width.
power consumption	The amount of energy consumed.
power save mode	An operational mode of mobile devices that can be activated by the user or the device itself to conserve energy.
power state	A user defined or predefined profile in regard to power consumption that can activated on a mobile device.
preferences	The general device or application configuration parameters that can be changed by the user.
pre-installed app	A mobile application that is installed by the device manufacturer. Usually the user is not able to de-install these applications.
QR code	QR code (abbreviated from Quick Response Code) is the trademark for a type of matrix barcode (or two-dimensional barcode).
remote device access	Interacting with a device physically located at a different location than its user, usually over the internet.
retain app data	User-generated data and or content of the app is retained on the device when the application is de-installed in order to be accessible by other apps or when the same app is reinstalled.
rooting	The process of gaining root access to the device operating system. Term is usually used on the Android platform. Similar to jailbreaking on iOS.
run-time environment	Implementation of the execution model. Also known as runtime system.
screen real-estate	The amount of space provided by the display.
software development kit (SDK)	A set of tools and libraries to develop software for a certain platform.
sensitive data	Data that needs special protection, such as passwords and personal data.
sensor	A device, module, or subsystem whose purpose is to detect events or changes in its environment and send the information to other electronics, frequently a computer processor.
session	A time-boxed event.
session sheet	A document to scope and record a test session.
side-loading	The loading/installing of an application via a means other than an application store.

Glossary Term	Definition
single-tier	A backend application design approach in which a single server provides all needed services for an application.
smartphone	A handheld personal computer with a mobile operating system and an integrated mobile broadband cellular network connection for voice, SMS (Short message service; often referred to as text), and internet data communication.
soft keyboard	A virtual keyboard realized in software, presented to the user on a display.
store-and-forward	A data synchronization approach where the data is stored locally and forwarded to the server when there is an appropriate network connection.
synchronous communication	A data transfer method in which data flows are sent (upstream) and received (downstream) at the same speed and is spaced by timing signals.
tablet	A type of mobile device, commonly used for devices with screens of 7”and larger.
thin client	In client/server applications, a client designed to be especially small so that the bulk of the data processing occurs on the server.
third-party marketplace	An app distribution platform not operated by a platform provider.
transaction-based apps	An application in which the user pays per transaction.
upload conflict	An error trying to upload a file which is already present at the upload destination.
viewport size	A virtual screen size used by the browser to adjust the layout to the screen.
Virtual Private Network (VPN)	An encrypted private channel via a public network.
wearable	A computer device worn on the body such as a watch or glasses.
web app	An application hosted on the internet used via a browser.
Xcode	An integrated development environment provided by Apple to develop OSX and iOS applications.

9. Index