ASTQB Certified

Mobile Tester

Version 2015

American Software Testing Qualifications Board

Copyright Notice
This document may be copied in its entirety, or extracts made, if the source is acknowledged.

This professional certification is not regulated by the following United Kingdom Regulators - Ofqual, Qualification in Wales, CCEA or SQA

Copyright © American Software Testing Qualifications Board (hereinafter ASTQB)

ASTQB Foundation Level Working Party – Mobile Syllabus: Judy McKay (chair), Randy Rice

Revision History

Version	Date	Remarks
Alpha	28 July 15	Alpha release for review
Beta	30 Aug 15	Incorporated Alpha release comments
GA	15 Sep 15	Incorporated Beta release comments

Table of Contents

Acknowledgements

This document was produced by a core team from the ASTQB Foundation Level Working Group – Mobile Syllabus:

Judy McKay (chair), Randy Rice.

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

The following persons participated in the reviewing, commenting and balloting of this syllabus:
Rex Black, Earl Burba, Jouni Jatyri, Pasi Kyllonen, Levente Nemeth, Andrew Pollner, Randy Rice, Gary Rueda, Szilárd Széll

0. Introduction to this Syllabus

0.1 Purpose of this Document

This syllabus forms the basis for the American Software Testing Qualification for the Mobile Tester. The ASTQB® provides this syllabus as follows:

1. To ISTQB® Member 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 training providers, to produce courseware and determine appropriate teaching methods.
3. To certification candidates, to prepare for the exam (as part of a training course or independently).
4. 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 ASTQB® may allow other entities to use this syllabus for other purposes, provided they seek and obtain prior written permission.

0.2 Examinable Learning Objectives

The Learning Objectives for each chapter are shown at the beginning of the chapter and are used to create the examination for achieving the Mobile Tester Certification. The Learning Objectives support the Business Outcomes.

1 Introduction to Mobile Testing - 75 mins

Keywords

Internet of Things, hybrid application, mobile application testing, mobile web application, native mobile application, wearables testing

Learning Objectives for Introduction to Mobile Testing

1.2 Expectations from Mobile Users

MOB-1.2.1 (K2) Explain the expectations for a mobile application user and how this affects test prioritization

1.3 Challenges for Testers

MOB-1.3.1 (K2) Explain the challenges testers encounter in mobile application testing and how the environments and skills must change to address those challenges

MOB-1.3.2 (K2) Summarize the different types of mobile applications

1.5 Equipment Requirements

MOB-1.5.1 (K2) Explain how equivalence partitioning can be used to select devices for testing

1.6 Lifecycle Models

MOB-1.6.1 (K2) Describe how some software development lifecycle models are more appropriate for mobile applications

1.1 What is a Mobile Application

Mobile applications generally fall into two categories, those developed specifically to be native mobile applications and those that were designed to be viewed through a web browser on a mobile device. From the user's viewpoint, there is no difference, although some browser-based applications may be optimized for the mobile device providing a richer (or at least more readable) user experience. From the developer's and tester's viewpoint, there are different challenges, goals and success criteria. This syllabus is focused on the applications specifically developed for use by a mobile device although there will be some discussion about applications that have become mobile despite the original intentions.

Mobile devices include any of the so-called hand-held devices including (dumb) mobile phones, smart phones and tablets/netbooks as well as devices that have been created for a specific use such as e-readers or a device used by a parcel delivery service that allows the driver to record delivery, the customer to sign and an image to be taken documenting the delivery. Mobile devices also extend to wearable items such as smart watches and glasses that allow access to specific applications and may include additional native functionality, such as telling time or improving vision. While some of the mobile application testing concepts discussed in this syllabus are applicable to wearable devices, wearable devices are not the focus of this syllabus.

The field of mobile devices is continually expanding as new uses are devised and devices are created to support those uses.

1.2 Expectations from Mobile Users

Mobile applications are becoming critical to daily living. Users expect 100% availability regardless of what they do to the device or the software. They expect usability that allows them to download and immediately use an application with no instructions or training. They expect exceptional response time, regardless of what else the device is doing and regardless of the strength or capability of the network.

Users have become impatient with slow software and have no tolerance for software that is difficult to use. Usability and performance testing have become vital to the release of any mobile application because of the expectations of the user, not necessarily because of the criticality of the functionality. This is different from traditional software where users are somewhat committed to using an application, even if it was a bit slow or awkward to use, particularly for enterprise software where the employee has no choice but to use it. If a mobile application is too slow or not attractive, the user will often have an option to download a different application. Organizations can lose customers if their mobile applications are not fast enough or pleasing enough. Competition in the mobile application industry is fierce, raising the importance of good testing and high quality products.

1.3 Challenges for Testers

Mobile users are everywhere and include everyone. Never before in the history of software has the user community been so vast or varied. Mobile uses vary from recreational to business-critical. Users expect seamless connectivity and instant access to information. The Internet of Things has put access into the hands of many but has also increased the expectation for all applications and devices to provide a consistent experience [InfoQ]. The Internet of Things includes many items that are not particularly mobile, such as refrigerators, or handheld devices, such as drones. While the Internet of Things is out of scope for this syllabus, it is important to remember that the experience people have with these devices affects their expectations for their mobile devices.

The set of mobile devices is continually growing. Software is expected to work, and work well, across a growing set of devices with constantly increasing capabilities, while providing an ever-expanding set of functionality to the novice and expert user.

1.3.1 Frequent Releases

One of the biggest challenges to testing is the frequency of the release cycles. Because the mobile market is so competitive, organizations race to be first-to-market with new features and capabilities. In order to meet these demands, support for development environments and tools has increased dramatically making the bar for entry into the market much lower than ever existed before. There are free development kits, free or inexpensive training and free distribution channels. This leads to many, many developers with the ability to quickly create and deploy an application. Testing has to adjust to the demands of time-to-market while also meeting the expectations of the users regarding functionality, usability and performance.

1.3.2 Portability/Compatibility

Although invisible to most users, there is an expectation that applications will work across devices and that devices will work together. There is an expectation to be able to easily and automatically transfer data between devices and to use the same applications from any device. The portability of an application is highly dependent on how it was developed and the deployment target.

The typical application types include the following:

- Traditional browser-based applications – The application is designed to work in a browser on a PC. It may or may not function well and provide adequate usability (e.g., scaling) when accessed from a mobile device.

- Mobile web sites – The application is hosted on the server but it is designed for mobile access across multiple compatible devices. Portability is a key concern.
- Mobile web applications – The application is developed for use by a variety of devices with the majority of the code residing on the web site. Mobile web applications must have communication with the web server in order to run. In some cases, the applications are the same as those used on the web site but generally a mobile version of the application is what is used by the mobile device.
- Native mobile application – A native mobile application is designed for a specific device family. These applications reside on the device and communicate directly with the device through the device's interfaces. Coding is normally done using tools designed specifically for the device. Testing a native mobile application requires either the specific device or simulators for that device and its software.
- Hybrid applications – Rather than being coded with the native device tools, these applications use a library or framework to handle platform specific differences. Device functionality is accessed via plug-ins that may be unique for different families of devices. Hybrid applications are designed to be more portable than native mobile applications but are still able to access unique device capabilities. Hybrid applications are often dependent on some level of connectivity with a web server and may also be subject to device/browser compatibility issues.

It is important for a tester to understand the intended target device(s) in order to know the portability requirements for testing.

1.4 Necessary Skills

Functional testing is required for mobile applications. The tester must have the skills necessary for manual functional testing tasks including requirements analysis, test design, test implementation, test execution, and results recording and reporting. These skills are covered in the ISTQB Foundation Level syllabus [ISTQB_FL_SYL].

In addition to the standard testing skills that are needed in any environment, mobile application testing also requires good capabilities for testing specific quality characteristics: security, usability, performance, portability/compatibility, and reliability. There are also new testing techniques, in addition to those covered in the ISTQB Foundation Level syllabus, that are applicable for mobile testing.

These quality characteristics, skills and techniques are covered in Chapter 3.

1.5 Equipment Requirements

Depending on the expected usage of the application, testing needs to cover representative devices. Representative devices are those whose behavior can be determined to be representative of other devices in the same class. For example, it might be determined that all iOS devices will behave the same way when running an application, therefore only one of those representative devices needs to be tested. The results from the test on one device is the same as would be seen if the same tests were run on another iOS device. This is equivalence partitioning applied at the device level.

Most devices will not fall into such large categories of similar behavior, so it is likely that a sample set of devices will be needed to determine compatibility for an application across devices. This usually results in having to acquire a large set of physical devices, use simulators, rent a lab full of devices or use alternate testing approaches. These options are discussed in more depth in Chapter 4.

It is important for the tester to approach any mobile testing project with a clear understanding of the equipment requirements. This is a key part in effective planning to determine the budget and

schedule and also requires the proper allocation of test cases across the various devices. Factors such as device location (rural, city), weather (sunny, rainy), usage location (indoors, outdoors), connectivity (WiFi, cellular) and others are significant in selecting the testing approach, people and locations.

1.6 Lifecycle Models

The requirements for fast development and deployment have pushed the software development lifecycle toward the iterative models, including Agile. Rapid prototyping is often used to quickly develop, gain feedback and successfully deploy a new product. Existing products are updated frequently and there is a tendency in the market to "push it out and let the users test it". This often results in frantic fixes being deployed to unhappy users.

As testers, we need to employ testing that will not substantially slow the progress of the product to market, but will help reduce the risk of a catastrophic failure. Risk-based testing approaches are critically important in the mobile application industry because there will never be enough time to test everything. The amount of risk and the matching amount of testing are correlated to the usage and criticality of the product. Smart phones, for example, have a wide variety of uses, some of them safety-critical. It is important to evaluate each application individually for its risk factors rather than to group a set of applications together since even though the functionality may be similar, the actual usage may determine the criticality. For example, if a viewing application should be able to display images at a certain resolution, not achieving that resolution might not matter for someone's holiday pictures, but could be safety-critical if those images are used by a remote doctor to analyze skin abnormalities to determine cancer treatment. Once a proper risk analysis has been conducted, the testing can be allocated to mitigate the risk to achieve the desired level of confidence in the released product.

Because many mobile applications are able to accept updates "over-the-air" (OTA), sending updates may be relatively easy and fast and it may be possible to force installation of the updates. Other mobile devices that have to be loaded from a central source (a PC for example) may not be as easy to update quickly if a significant defect is found. The ability and ease with which updates can be applied may be a factor in determine release risk. It is also a factor in determining how much effort will be needed for maintenance testing.

Many products are developed incrementally. An initial, simple version of the application is developed and deployed. Features are then added incrementally as they become ready and as the market demands. This type of development allows the product to be introduced quickly without compromising quality while additional features are developed internally with testing time allocated.

Sequential lifecycle models (e.g., V-model, waterfall) are used less frequently for mobile applications due to the need to get a product to market quickly. Documentation tends to be minimal and testing tends to follow more lightweight methods with less documentation. Safety-critical applications still tend to follow sequential models as do other applications that are under regulatory control.

Testing approaches are discussed in Chapter 2.

2 Test Planning and Design – 60 mins.

Keywords

minimal essential test strategy, operational profiles, risk analysis

Learning Objectives for Test Planning and Design

2.1 Identify Functions and Attributes

MOB-2.1.1 (K2) Explain why use cases are a good source of testing requirements for mobile applications

2.2 Identify and Assess Risks

MOB-2.2.1 (K2) Describe different approaches to risk analysis

2.3 Determine Coverage Goals

MOB-2.3.1 (K2) Explain how coverage goals will influence the level and type of testing to be conducted

2.5 Identify Test Conditions and Set Scope

MOB-2.5.1 (K2) Describe how test analysts should take the device and application into consideration when creating test conditions

2.1 Identify Functions and Attributes

Feature rich mobile devices are difficult to test. It is important to focus on the functions and attributes that are within scope for the testing effort. For example, if the goal is to release a new application across multiple smart phones, the focus will be on the capabilities of the application, the interaction of that application with the device and the quality characteristics that are important for the success of the application (i.e., usability and performance). If the project is to release a new smart phone, the scope is different. In this case the tester will focus on the capabilities of the phone itself, its ability to support a sample of applications, communication between the device and the network (also WiFi and other forms of communication such as IP-over-USB), and various other quality characteristics. The focus of this syllabus is testing the mobile applications rather than the device itself.

Requirements tend to be brief. There may be a specification, a requirements document, use cases or user stories. In general, the tester should not expect comprehensive requirements and should instead plan to work at the use case level where usage scenarios are identified. If the use cases are not available, the tester should seek them out to understand the expected usage and to focus the testing accordingly.

In order to scope the testing, it is important for the tester to understand the attributes of the application that are important to the user and prioritize them appropriately. If security and performance are more important than usability, this will help to identify the risks and determine the amount and type of testing that will be needed in each area. The stakeholders must understand that each attribute desired to be tested will require an investment in people (with the appropriate skills), tools and environments.

2.2 Identify and Assess Risks

A mobile application project that is not safety-critical or mission-critical is usually characterized as being feature-rich but time-poor, meaning that there are many features, but little time for implementation and testing. Requirements tend to be brief and informal. As a result, when identifying and assessing risks, it is important to use a lightweight process. One way to approach the risk analysis is to think of the application in two ways, the physical and the functional.

For the physical capabilities consider the items that are physically touched by the user (e.g., buttons, icons, display, graphics) and physical features of the device that are used by the user (e.g., rotation, accelerometer). These capabilities enable the functionality of the application but are not functions themselves. Once these items are identified, create a grid that lists the category of the item (e.g., display) and list the capabilities that are of critical, high, medium, and low importance to the user. For example, it might be critical that an image loads completely in normal situations, high importance that the resolution is acceptable, medium importance that it loads consistently without retries, and low importance that it retries the load if the connection is dropped. Similarly, it might be of critical importance that an image rotates when the device is rotated, high importance that it resizes upon rotation, and medium importance that the text rotates with the image.

For the functional capabilities consider the features of the software (e.g., accurate map loading for a navigation application). In this case, it might be critically important that the correct map is loaded, highly important that the map shows the car's location, but only of medium importance that the map shows fuel statements, and low importance that it shows construction sites.

This lighter-weight approach allows the tester to understand the physical aspects of the device that will need to be tested, either on a real device or a simulator, as well as the features that are important to the user. By working through a spreadsheet of this type, the tester can find requirements that might not have been stated and can help discover features that are implemented but not documented.

Examples of lightweight approaches to risk analysis are available from multiple sources. Traditional risk analysis approaches can also be used in a lighter-weight fashion to better fit mobile testing. See [Paskal] for information regarding the Minimal Essential Test Strategy (METS), [Black09] for a discussion of risk-based testing, and [vanVeenendahl12] for a discussion of the PRISMA® approach.

It is important for the tester to adapt the risk identification and assessment process to fit within the timelines of the project. Heavyweight methods will not be successful in this environment and will tend to delay the testing.

It may also be useful to consider production metrics when defining risk areas. For example, the following metrics could be used [Webtrends]:

- Total downloads – Indicates the amount of interest in the application and provides the upper bound for the maximum number of concurrent active users.
- Application users – Indicates how many people actually use the application (not just downloaded it).
- Active user rate – Provides the ratio of the number of application users to the total number of downloads.
- New users – Provides the number of users who first used the application within a period of time (particularly interesting when compared to the attrition rate that can be derived from the active user rate).
- Frequency of visit – Provides the ratio of the number of visits to the number of users over a period of time (can be used to gauge user loyalty).
- Depth of visit – Indicates the number of screens viewed during the average visit.
- Duration – Indicates the average amount of time spent in the application.

- Bounce rate – Provides a ratio of the number of user visits that had only a single view (people who downloaded the application, tried it, and then never used it again).

These metrics can be used to identify high risk areas that can be addressed by testing or development. For example, a high bounce rate may indicate usability issues. The active user rates can be used to develop realistic performance testing goals.

2.3 Determine Coverage Goals

Once the risks have been identified and assessed, the coverage goals must be determined. While the tester will need input from others, such as the test manager, it is important to consider all the areas to be tested and get agreement with the team that the coverage goals are realistic and will accomplish the testing goals for the project.

The following areas should be considered and the desired coverage determined before starting testing on the project:

- Requirements – If there are requirements, requirements coverage should be used as one of the testing guidelines. Traceability from the tests back to the requirements is useful because requirements for mobile applications often change as new features are added and existing features are updated or modified. The traceability will help the tester know which test cases need to re-executed when changes occur.
- Risks – The identified risks must be addressed by testing, and traceability may be needed between the test cases and the risk items.
- Functions – The capabilities of the software will be tested but should also be tested in accordance with the risk associated with each. A complete list of functions will help to set the risk levels as well as to track coverage of each of these items.
- Code – Because of the speed of the development of mobile applications, unit testing is very important and code coverage goals should be stated before development starts. Automated unit testing, particularly when employed with continuous integration and deployment, will help to improve the quality of future updates as the same tests can be run each time without significant manual time and effort. Fault metrics and technical debt measures can be used to track the quality of the software.
- Devices – Coverage across devices must be known at the beginning of the project so those devices can be procured or simulators can be bought or built. The developers must provide input regarding the expected variability between devices so intelligent decisions can be made regarding which device behavior can be determined to be representative. Device-based application testing is usually prioritized based on the expected usage of particular devices with particular applications. Since it will not be possible to execute all test cases on all devices (and the permutations of those devices), allocating the test cases across the supported device configurations is an important risk mitigation activity.
- Connectivity – Coverage must include the way in which a device connects to the Internet (including cellular, WiFi, Ethernet, and in some cases the ability to switch). This should also include access to any additional services (such as loading style sheets) and potential side-effects of network issues such as latency, jitter and re-tries.
- Geography – The geographic location of expected use can influence the testing. If an application is expected to be used only at high altitudes, the test environment will need to take that into account. Devices that must respond to intermittent or slow networks will be tested differently from those that will only be used in offices with highly reliable, fast networks.
- User Perspectives – Designing good test cases requires a knowledge of the users including their expectations, knowledge, capabilities, personas, and operational profiles (what they will be doing). Testing will need to simulate usage by the various expected users.

Understanding the coverage requirements for testing is important for setting the scope and timelines of the testing effort as well as to help determine the types of equipment and environments that will be needed.

2.4 Determine Test Approach

Once the coverage goals are determined, the proper test approach can be decided. The test approach must consider the following:

- Environments – The tests must be conducted in certain environments and those environments may also have associated conditions (e.g., outdoors while raining).
- People – The product is intended for certain user types. The actions of those users must be built into tests including any variability based on the user (e.g., someone with poor eyesight may always zoom images to view them).
- Industry context – The target industry can influence the required test approach (e.g., safety-critical, mission-critical, COTS, games, business applications, social network).
- Schedules – The reality of the schedule must be considered when determining the test approach, with the highest priority (highest risk) tests being conducted first.
- Scope – The testing scope must be limited and clearly stated to set the expectations for the coverage to be achieved and the risk mitigation goals.
- Evaluation – Evaluation of test results tends to be different for mobile projects because much of the non-safety-critical testing is done with less structured techniques and with simulators and emulators. The evaluation method must be clearly stated and understood by the team members so they will understand test status reports and the final test summary report.
- Methods – Testing methods vary for mobile projects. These are discussed in Chapters 3 and 4 regarding specific quality characteristics, environments and tools.

Depending upon the formality and criticality of the project, the test approach may be documented in a traditional test plan or may be informally documented in a brief project document. Either way, the approach should be documented because agreement to the approach is critical within the project team.

2.5 Identify Test Conditions and Set Scope

The test conditions are the building blocks of the testing to be conducted in a mobile application project. Time to create test cases may not exist in a fast-paced project. In this case, identifying the test conditions, assigning risk-based priorities to each and conducting testing to address each of identified condition may be the most efficient method for testing within the limited timeframe.

Test conditions consist of the physical capabilities of the software within the device (e.g., buttons, icons, screen zooming, device rotation, geolocation), the functionality of the application (e.g., displaying an image, displaying a map, accessing a bank balance) and the non-functional areas such as performance and usability. Each of these capabilities and features has a number of conditions that should be tested. Using the risk assessment, these conditions can be prioritized for testing and the scope of testing can be set. For example, if the application will access banking information, the application may have a login capability. To test this login, the tester needs to test a valid username/password, invalid username/valid password, valid username/invalid password, and so forth. Each of these combinations is a test condition. Since there can be many test conditions for a single feature of the software, it is important to identify the critical and high risk conditions to be sure those are tested. The low risk items may be left untested or may be tested as part of other tests.

Identifying and prioritizing the test conditions sets the scope for the testing. With limited time, priority/risk-based testing will ensure the most important items are tested to some level of coverage. When time runs out and the coverage is deemed sufficient, testing is complete.

2.6 Regression Testing

Regression testing for mobile applications is particularly challenging. Not only does the software change rapidly (including the firmware), but the devices are continually changing as well. The more devices supported, the larger the set of changes. Regression testing should be conducted regularly for mobile applications, even if the application itself has not changed. As discussed in this syllabus, test automation and access to device labs and simulators is critical to a successful mobile application project and are required for a good regression test practice. When regression testing is automated and devices are available (via labs or simulators), the regression testing can be scheduled to run at regular intervals such as once a week. This does require that the test devices and simulators are also being updated regularly so the regression testing is reflecting the functionality of the software on the target devices.

3 Quality Characteristics for Mobile Testing - 290 mins

Keywords

geolocation, Teststorming™

Learning Objectives for Quality Characteristics for Mobile Testing

3.2 Functional Testing

MOB-3.2.1 (K3) For a given mobile testing project apply the appropriate test design techniques
MOB-3.2.2 (K1) Recall the purpose of testing for the correctness of an application
MOB-3.2.3 (K2) Explain the important considerations for planning security testing for a mobile application
MOB-3.2.4 (K2) Summarize the concepts of perspectives and personas for use in mobile application testing
MOB-3.2.5 (K2) Summarize how device differences may affect testing
MOB-3.2.6 (K2) Explain the use of Teststorming for deriving test conditions

3.3 Non-Functional Testing

MOB-3.3.1 (K3) Create a test approach that would achieve stated performance testing goals
MOB-3.3.2 (K1) Recall aspects of the application that should be tested during performance testing
MOB-3.3.3 (K2) Explain why real devices are needed when simulators are used for testing
MOB-3.3.4 (K3) For a given mobile testing project, select the appropriate criteria to be verified with usability testing
MOB-3.3.5 (K2) Explain the challenges for portability and reliability testing mobile applications

3.1 Introduction

Mobile applications, similar to other applications, have functional and non-functional quality characteristics that must be tested. While all the quality characteristics mentioned in the Foundation syllabus [ISTQB_FL_SYL] are applicable, this syllabus covers those that are particularly important in the mobile application testing scope. While not all of these are applicable to every mobile application, each should be considered to ensure that nothing is skipped and to ensure testing is prioritized correctly.

3.2 Functional Testing

3.2.1 Introduction

Functional testing is designed to assess the ability of the application to provide the proper functionality to the user. It tests what the software does. For mobile applications, functional testing covers the following:

- Correctness (suitability, accuracy)
- Security
- Interoperability

Each of these is discussed in the sections below.

3.2.2 Correctness

Correctness testing is done to ensure the software provides the right functionality in a way that works for the user (suitability) and that the functionality is provided correctly including all data delivery (accuracy). If the capability is there, but it is not delivered in a suitable way, the product may be unusable. For example, if a smart phone application cannot scale an image down to fit on the screen, it is not suitable. If it can scale the image, but it is the wrong image, it is not accurate.

3.2.3 Security

While security testing is best left in the hands of the security experts, all testers should have some awareness of security vulnerabilities and areas that should be covered by testing. Some tools are available that can help with security testing, such as static and dynamic analysis tools, but good security testing requires a current knowledge of security issues, testing methods and tools, and the technical ability to create security tests (which often involve coding).

3.2.3.1 Security in Mobile Testing

Security in mobile applications poses more threats than traditional applications. The following should be considered when planning security testing or considering what should be tested:

- Mobile applications are generally more easily attacked by hackers than traditional applications. This is partly due to the lengthy communications over public networks and partly due to the tendency for users to download many potentially vulnerable applications which can expose other applications residing on the device.
- People are too trusting. They tend to download applications without concern although they would never open an email attachment from someone unknown. A great deal of personal information is kept on mobile devices such as smart phones and tablets because they are convenient and always available. Rather than recording passwords on a sticky note on a desk, passwords are often recorded in the notepad application on the device itself.
- Devices get left behind. People misplace mobile devices frequently. This leaves the device open to tampering, particularly when it is protected by a single short password or pattern.
- Mobile devices are often donated, sold or traded-in without the existing data being wiped. This provides a rich opportunity for the recipient to access all types of user data – passwords, user names, pictures, videos, contact information (e.g., name, phone, e-mail).

An important part of mobile application development is to compensate for the lack of security knowledge on the part of the user. An important part of testing mobile applications is to ensure that the security is in place and is working correctly. Testers need to make sure sensitive information, such as passwords or account information, is not stored unprotected on the device. While malware (hostile or intrusive software) will always exist, the application should protect itself from attack and the device itself should have some protection to validate installed applications.

While it changes year by year, the following is the list of the top 10 mobile risks in 2014 according to [OWASP]:

- Weak server side controls
- Insecure data storage
- Insufficient transport layer protection
- Unintended data leakage
- Poor authorization and authentication
- Broken cryptography
- Client side injection
- Security decisions via untrusted inputs
- Improper session handling
- Lack of binary protections

Knowing the security risks helps the tester know what to test and can help as a reminder for developers when they are coding.

3.2.3.2 Security Testing Approaches

Because security testing is often carried out by security testing experts, the tester may not need to know how to set up and complete security testing. It is however important for a tester to ensure they are covering the basics during their testing. This includes testing the following:

- Access – Ensure the right people and applications have access and those without permissions are denied access. Also ensure that access is limited to only the functions and data that the user should be able to access. This is similar to access security for any type of application.
- Protecting data while on the device – When data is stored to the device, it must be secure. This means that such information as passwords, account information, credit cards and so forth that are used in transactions are not stored in an accessible format on the device even during the transaction.
- Protecting data that is in transit – Information is passed between the device and servers via the network (e.g., cellular, WiFi) and information that is passed between devices (e.g., WiFi, Bluetooth, SMS). Any data that should be secured must be encrypted during this transition to protect it from interception and misuse. This includes transactions that may encounter errors or have to be retried.
- Policy-based security – Organizations may have security policies that indicate how data is handled and who may access it. When data is being transferred to/from a device or stored on a device, these policies apply just as they would with a non-mobile application.

As with any testing, it is important to understand the application and its uses. Security for a banking application will not be the same as that used for a memory game.

3.2.4 Interoperability

Mobile applications must be tested for interoperability to ensure they interact properly with other components, devices and systems. Mobile applications must be able to exchange information and images with other software. For example, an image captured by the camera can be sent via email on a smart phone.

Testing for interoperability is highly dependent on the capabilities and interactions of the application being tested. At a minimum, an application likely transfers data back to a web server that maintains a storage of information (e.g., highest score achieved in a game, the current weather forecast). It is not unusual for applications to act alone and with other applications loaded on the same device. Since new applications may be added at any time to a device, trying to test for the superset of interacting applications will likely lead to frustration. This is why the risk-based approach using a lightweight means to capture this information is a good way to approach the problem of too much to test in too short a time period.

Interoperability testing can be expanded into verifying compatibility of the application across environments. An application may need to work on a variety of devices operating at different speeds. This form of interoperability testing is sometimes called compatibility testing. One factor that makes compatibility testing of mobile applications and mobile web sites so challenging is the number of browsers and versions supported by each application and device brand/type. It is important to know the list of devices on which the application is intended for use so a reasonable testing matrix can be developed and the testing can be divided between the devices using techniques such as the combinatorial testing techniques. Sources of configuration information include web server logs, web analytics and store analytics (such as iTunes and Google Play) to see which percentage of users for a particular application use a particular device/browser.

3.2.4.1 Device Specific Considerations

Mobile devices are varied and have a wide range of capabilities. One of the factors in interoperability testing is understanding the commonalities and differences between devices. Specific versions of devices may have different capabilities that may affect an application's capabilities and usability. For example, an application running on a slower device (or one with less memory) may exhibit different characteristics than one running on a fast device. An anti-glare screen protector may render certain user interface designs difficult to read and/or access. This device specific information tends to change rapidly and should be determined immediately prior to testing. It may also be necessary to anticipate new device features so that testing can occur before the features are widely available (e.g., higher speed, bigger memory, operating system requirements).

3.2.4.2 Testing Peripherals

It is also important to remember that devices may have peripherals attached or built in such as scanners, card readers, bio recognition equipment (e.g., fingerprints scanners), cameras, altimeters, microphones, speakers, and so forth. If the application may use a peripheral or may be affected by the presence or absence of a peripheral, the application must be tested both with and without the peripheral. This is a consideration if simulators will be used for testing since simulators for peripherals may be needed. In the case of peripherals though, actual device testing is usually required to some degree.

3.2.4.3 Device Differences

Mobile devices have many differences, even within the same type of device such as a tablet. For example, communication protocols may be different, transmissions may or may not be secured, the device may have the capability to be docked and transfer information. A device may be able to recognize other devices of its type when they are within a certain geographic area. The variability between devices and the commonalities they share all introduce testing opportunities. It's important to understand how an application will interact with a device or set of devices and how differences in those devices may affect the application. For example, a device may share geolocation information with the application which then shares it with other applications and allows communication to other applications that are running on similar devices in the same area. If the geolocation information is secured on one device, but not on another, what will happen?

As devices add more and more features and more devices enter the market, these differences will become a larger factor in testing.

3.2.5 Test Design

When designing the tests for a mobile application, the following should be considered:

- Functionality of the application
- Functionality of the device
- Risk within the subject domain of the application (e.g., a mobile device used to deliver medical information to ambulances)
- Network connectivity
- Operating systems
- Power consumption/battery life
- Type of application (native, hybrid, etc.)

The functionality of the application can be determined from the requirements, use cases, specifications or even conducting exploratory testing to learn about the application. The functionality of a device, particularly if the device is made by another organization, must be determined by reading the published specifications, experimenting with the device or from talking with others who are familiar with the device. Designing tests for a mobile application requires considering both the features of the application to be tested as well as the capabilities of the device.

The capabilities and features of the target device must be understood, particularly if those capabilities will be utilized by the application. The following is a list of some of these capabilities, but remember the list is always expanding:

- Screen size and resolution for display
- Geolocation (ability to detect the device's geographic location)
- Telephony (ability to act as a telephone)
- Accelerometer (senses acceleration on three axes – up/down, side to side, back and forth – used for games and orientation)
- Gyroscope (senses orientation based on angular momentum)
- Magnetometer (measures the direction of magnetic fields – can act as a compass)

If the application depends on using a magnetometer for example, the test design must include tests for devices with various types of magnetometers as well as for devices without the capability.

In addition to the functionality, test design also needs to include installation of the application. Many applications can be installed over-the-air (OTA) which means that testing needs to include interruptions at any point in the installation, re-installation, upgrades and de-installation. Permissions and payment may also be required to install applications and so must also be tested.

The risk of the application is assessed by the means discussed in section 2.2, Identify and Assess Risk.

3.2.5.1 Using Core Foundation Techniques

The standard black-box test design techniques are explained in the ISTQB Foundation syllabus [ISTQB_FL_SYL] and further developed in the ISTQB Advanced Test Analyst syllabus [ISTQB_ATA_SYL]. These techniques are applicable for mobile application testing and are valuable in testing both applications and devices. Use of these techniques will help the tester ensure that the desired test coverage is achieved. These techniques are briefly summarized here:

- Equivalence partitioning (EP) – Determine equivalences classes based on equivalent processing and test one item from each class assuming the results for the one item are representative of the entire class. For example, assume all cameras with the same megapixel capabilities will create an image of the same quality.
- Boundary value analysis (BVA) – Select tests based on the boundaries of ranges of inputs or outputs. For example, test the maximum number of names that can be stored in a contact list, test maximum + 1, test one and test zero.
- Decision tables – Test combinations of inputs and/or stimuli (causes) with their associated outputs and/or actions (effects). For example, test that an incoming email results in the configured sound.
- State transition models – Test the transitions between two states of a component or system. For example, test that the display changes from bright to dim when the exterior light changes.
- Use cases – Test the primary (main) scenario and all alternate scenarios. For example, test that a delivery driver can note that they delivered a package, get a signature and record the location of the delivery.
- Experience-based techniques
    - Exploratory testing – Test by simultaneously designing and executing tests while learning about the application. For example, for a new application, test it by using it to accomplish a single task and document any defects found.
    - Attacks – Test by targeting specific expected faults in the software. For example, target communication security.
- Defect-based techniques – based on a defect taxonomy, target specific defect types for testing. For example, test handling of invalid inputs.

- Combinatorial techniques – Test across different combinations of characteristics based on the information supplied by the model. For example, use pairwise testing to determine the combinations of devices and device features on which to test the application.

Some of these techniques, particularly state transition models, are well-suited to testing interactions between the device and the software. Others, such as equivalence partitioning and the combinatorial techniques help to reduce the test set down to a manageable size. Experience-based and use case testing help focus the tester toward real world usage and scenarios

3.2.5.2 Using Mobile Specific Techniques

In addition to the techniques in the section above, there are also techniques that are commonly used in testing mobile applications. There are some overlaps between this list and the list above, but the best combination of techniques to use should be determined by the features of the application, the capabilities of the device that interact with the application, the criticality of the application, the time available, and the skills/knowledge of the tester.

The following techniques are commonly used in mobile application testing:

Session-based – these testing sessions are designed to be uninterrupted from start to end, reviewable by other testers or managers and chartered to ensure the focus matches the goals of the testing.

Exploratory testing can be expanded into the concept of exploring different user perspectives. Because the user base is so varied, it's important to consider the different perspectives those users bring to their usage of the device. The goal is to simulate real usage and concentrate on specific aspects of the software and its interaction with the device [Whittaker]. These perspectives and usage scenarios should cover the following:

- Skill level of the user (see persona-based testing below)
- Location of the user (e.g., indoors, outdoors, home, work, in a car, on a plane)
- Lighting in the environment (e.g., dark, bright sunlight)
- Weather conditions (e.g., rain, wind)
- Connectivity (e.g., strong, weak, intermittent)
- Accessories available (e.g., interaction with each)
- Motion (e.g., stable, in hand while walking)

Scenario-based testing, similar to use case testing, tests the paths that a user is likely to follow to perform a defined task. The validity of the scenario directly influences the effectiveness of the test. Testing scenarios that a user will not follow can result in wasted time that could be better spent testing frequently used paths.

Because the user base is so varied, it is important to consider the different types of people that will be using the software. People vary widely in skills, capabilities, and needs. The following is a list of some of the personas that can be used for persona-based testing:

- First time user
- Casual user
- Frequent user
- Expert user
- Confused user (does not understand any of the software)
- Angry user
- Frightened user (afraid of technology)
- Impatient user
- Malicious user
- Playing user (experimenting with the software)
- Technically knowledgeable user
- Age class user (e.g., over 65)

Persona-based testing requires that the tester understands the viewpoint of the user and is able to interact with the software as that user. This can be a difficult task, particularly when the tester's level of knowledge is much higher than the user persona being tested. Good usability requirements will help to cover the software characteristics that are needed to accommodate the various user personas.

Because device and application usage can be so varied, it is often helpful to approach testing from a viewpoint that is independent from the requirements. For example, Teststorming™ [Rice] may be used to derive test cases and scenarios via brainstorming or using mindmaps. This technique helps the tester to think of new and creative uses of the software that might not have been considered during the design and development of the product. As the usage and capabilities of mobile devices continues to expand, test case creation requires forward thinking, beyond what is stated in the requirements.

Reminder checklists are helpful when testing mobile applications to help the tester ensure that all aspects of the software are being addressed by testing. Since requirements for mobile applications tend to be lightweight, the tester cannot rely on these documents as the only guide for testing.

3.3 Non-Functional Testing

Non-functional testing concentrates on how functionality is delivered to the user. For a more complete discussion on non-functional testing, see [ISTQB_FL_SYL]. This section concentrates on the non-functional quality characteristics that are of primary importance in testing mobile applications.

3.3.1 Performance Testing

In general, performance testing is verifying the response time of the system when it is experiencing a defined load. Performance testing also considers throughput and resource utilization. With mobile applications, there are more considerations for performance testing such as network connection type and strength, device type, device memory and other conditions that may be difficult to control. Mobile applications are used for a wide variety of capabilities, but some applications have higher performance requirements than others. For example, GPS location applications that may be using the geographical position of the device to transmit driving directions have critical performance requirements in order to be able to adjust for changes to the planned course. Applications that are used for time-critical transactions, such as stock trading, also have stringent performance requirements. Applications that provide media have requirements to provide consistent performance that is sufficient to provide a good user experience without problems such as pausing during video display.

3.3.1.1 Performance Testing for Mobile Software

In addition to the normal performance testing that should be conducted regarding the server's ability to handle traffic [see ISTQB_FL_SYL], there are other considerations for mobile application testing. Network connectivity plays an important part in the performance of a mobile device. Variables such as connection speeds, time required to connect and reconnect if disconnected and network latencies are all factors in the ability of the application to deliver the desired performance. Unreliable or inconsistent network connectivity may result in multiple re-tries or the application trying to proceed with some data loss. This can be particularly critical if the loss is from information the application needs to function.

Performance testing a mobile application starts with ensuring that the application itself, and its interactions with the server, are as efficient as possible. A slow application or slow response from the server will only get worse when the application is running on a mobile device. In addition to the normal tests, mobile application performance testing should also cover the following aspects for the application itself:

- Application launch time – This includes the time from the user's first indication that they want to use the application until the time the application is fully usable.

- User interface delays – This is the time that is spent between receiving a user interaction (e.g., pushing a button, moving an image) until the time the application provides the response to the user.
- Irregular performance – This is a problem when the performance is noticeably varied for the same type of transaction.
- Visual indicators – As with any user interface, the user must be given an indicator that processing is occurring and there is a wait. Since these indicators usually appear after a certain length of waiting time has occurred, it is important to test to ensure that the indicator appears correctly even when performance is degraded.
- Resource usage – A mobile application is likely running in a shared environment. This makes efficient usage of CPU, memory and battery important not just for the application but for the overall device as well. Mobile devices are rarely running just one application. Performance must be verified with an expected set of other applications and processes running. This will help the verify what the real user experience will be when running the application under test.
- Task completion – This is an overlap between usability and performance, but it is important to test the time it takes for a user to complete tasks that they would expect to accomplish with the application.
- Code inefficiencies – While the code contained in a mobile application may be compact, it is still possible for the code to have bottlenecks, endless loops or other inefficiencies that impact overall performance. These inefficiencies can be detected by static analysis (reviews and/or tools) and well as dynamic analysis (tool-driven).

In addition to these, web applications also need to be tested for:

- Site page loading time – Since a mobile application may be running from a web site, the time it takes to load the site pages can have a significant impact on the user's experience with the application.
- Delays – Delays can occur for many reasons, particularly those related to server and network time.
- Resource usage – While a web application will take less memory and CPU from the device, it may require more network bandwidth from the device because of the larger amount of data being passed back and forth from the server.

3.3.1.2 Performance Testing Approaches

Identifying valid personas is important for both performance and usability tests. Personas define the characteristics of the user as well as the tasks the user is trying to accomplish. By using this information, scripts can be built that can simulate typical user transactions. These personas can be duplicated by automated tools to provide multiple virtual users who will interact with the system through a device in a prescribed way. Since devices may be difficult to procure, device simulators are often used to supply the same interaction as the device without requiring the actual device to be present during the testing.

Performance goals and testability goals must be established when the application is being designed and the target environments are being specified. These goals can then be used to compare against the actual results of the testing. Performance testing can start as soon as individual components are available. By building and executing performance tests as the application is being developed, particularly in a continuous integration environment, poorly performing components can quickly be identified.

The environment will always have an impact on the application's performance. A device that is inherently slow due to poor design, high overhead, slow communication, or any other factor will cause the application to appear to perform slowly as well. When testing with simulated devices, it is important to include a set of real devices to ensure the simulated performance is reflective of the performance that will be experienced by a real user on a real device. It is also important to understand what is being tested. A mobile application that is deployed on a user's device has different

performance concerns than a web application that is accessed from a mobile device. Understanding these differences before designing performance tests will help ensure valid results and the proper focus of the testing.

Overall poor performance may cause people to abandon the application. What is "slow" is determined by the user, and user expectations are continually being refined so that yesterday's fast performance may be unacceptably slow tomorrow.

3.3.2 Usability Testing

Mobile applications tend to have a wider and more varied user base than traditional applications. This is partly due to the ease of access to these applications and partly due to general acceptance that applications should be available anywhere, anytime. This poses significant usability concerns, particularly on the part of those planning and conducting usability tests.

3.3.2.1 Usability Testing for Mobile Software

In addition to traditional usability areas such as navigation, colors, sounds, accessibility, and others, mobile applications have some additional areas for testing. These include:

- Simplicity – Mobile applications must be designed for simplicity and ease of use.
- Layout – Interaction with mobile applications is sometimes done via a device such as an electronic pointer or pen, but many devices require the use of fingers. This means the application must allow space for fingers and perhaps for displaying a keyboard that is large enough to be used for text entry.
- Intuitiveness – Users expect to be able to load a mobile application and immediately use it. They may experiment with it for a bit, but they will quickly decide if it is intuitive to use or just too difficult. If it is too difficult, most users will discard the application and look for another. If instructions will be displayed to the user, they must be visible, but not intrusive.
- Navigation – While navigation is a concern with traditional applications as well, it is even more important for mobile applications. The user has an expectation to be led in the direction they need to go rather than having to determine their path from a list of many options. Mobile applications are expected to be simple and easy to navigate.

Mobile applications, more than traditional applications, will experience a high rate of abandonment if they are not considered usable by their users.

3.3.2.2 Usability Testing Approaches

As was mentioned in performance testing, personas are needed for usability testing to ensure testing is covering a representative set of user types. It is important to remember that users are not just end users. Applications are sometimes used to increase sales for a company. If a particular campaign has been used across the mobile applications for a particular company, that company's sales team may need to see metrics regarding number of downloads, number of responses, and other information.

User expectations are an important consideration. This is an area where expectations can be expected to change as more applications become available, new and improved devices are introduced and speed is improved. Usability experts will be challenged to stay current with market expectations for product usability.

When approaching usability design and testing, real users are needed. Observing users actually using the application will help target the testing to cover real usage scenarios and may also highlight areas where the interface is confusing or navigation is unclear. If possible, obtaining user feedback is also helpful to understand what they like about the application and to identify areas where improvement would be helpful. Usability labs and surveys may be used to help accomplish obtaining this information in a controlled environment.

In addition, simple metrics can help bring objectivity to the subjective topic of usability. Measuring items such as the following can be helpful in understanding usability factors:

- The number of tasks fully completed
- The time taken to perform a complete task
- The mistakes made in performing a task or series of tasks
- The number of clicks (actions) needed to perform a task

These measures can also be obtained by first, second and third attempts which can then be used to determine the learnability of the application.

Effective usability testing can be conducted with a small set of representative users [Nielsen]. If the users fit the identified personas, they can provide valuable feedback relatively inexpensively that can be used to improve the application and improve future designs.

With mobile devices being used by so many people, accessibility must be considered during testing. If compliance to particular accessibility standards is required, it is important to obtain tools that can scan the application for compliance or to conduct the manual testing necessary to ensure compliance. Accessibility considerations include such items as readability, color usage, sound usage, human interaction such as typing, ability to resize the screen or change contrast, and so forth. It is also important to consider accessibility in terms of environmental challenges such as bright sunlight, darkness, rain and so forth.

Mobile devices are feature rich and this feature set is sometimes dependent on accessories to the base mobile device. Accessories include cameras, scanners, credit card readers, headphones, keyboards, and other devices that can be built into or attached to the base device. When testing an application, it is important to consider any accessories that might interact with the application or the environment shared by the application. Consideration must be given to concurrent processing when using accessories. For example, a credit card reader might supply input to a banking application running on the phone, but that card reader might not work when the camera is also in use. These types of interactions between accessories must be considered when selecting test device configurations.

3.3.3 Portability Testing

Portability testing focuses on how well an application will function when moved into a target environment. Good portability testing requires a good understanding of the target environments and the characteristics of those environments.

3.3.3.1 Portability Testing for Mobile Software

Mobile software is intended to run on a mobile device. Devices are plentiful and the numbers, types and capabilities continue to increase. Mobile testing is usually concentrated on a subset of representative devices intending to cover the most common environments and environment variables that could affect the application under test. The key to good portability is a good design. It is important for the tester to work with the developer in determining areas to be tested due to device differences. For example, a developer is usually aware of modifications that were required for an application to work on both iOS and Android devices. That said, developers will sometimes miss nuances between devices and between different versions of similar devices. Obtaining a good set of representative devices can be split between procuring the actual devices and using accurate simulators for some devices.

3.3.3.2 Portability Testing Approaches

Users do not usually have an awareness that an application may have to be modified to work on different devices. This results in an expectation from the user that a mobile application will work on a set of devices and that they can expect the same level of usability from the application on a smart phone, a tablet and a PC browser. This may not be a realistic expectation, but it is often a tester's job to verify which environments enable the application to work and work well.

Portability testing for the present requires testing across a known set of devices and software versions. Portability testing for the future requires anticipating which devices will still be popular, what potentially conflicting or limiting features they may have and how a user will expect to use them. Because the field of mobile devices changes so rapidly, it's important for a tester to understand the new devices and the planned release dates for the major new changes. Procuring updated hardware devices can be expensive but simulators tend to lag behind the introduction of the physical device. As a result, sometimes the market is testing an existing application on a new device before the test team has a chance to try it.

Developers must do their best to future-proof their designs. Device manufacturers will normally maintain a level of backward compatibility (meaning that software that ran on the previous versions will still run on the new version), but there is a short life span in mobile devices and old releases will fall out of support more rapidly than is seen with desktop software.

Portability testing must consider whether the application is using a native device interface or a more portable interface. An application using a native device interface will likely encounter portability problems when the application is installed and run on a different type of device. An application with a more generic interface is designed to be ported to different devices.

Software that has been ported to another environment will sometimes exhibit performance and usability differences in the new environment. Any porting project must consider additional performance and usability testing to ensure an acceptable level is still achieved. For example, a mobile application that displays flight information might be quite readable on a tablet, but might not scale correctly or allow resizing on a smart phone. Similarly, software that was written to be fast on a specific device may be quite slow on a different device just because it is not optimized for that environment.

3.3.4 Reliability Testing

Because mobile software is everywhere and in many hands, it has become an important part of both business and personal life. As people become more and more dependent on it, reliability becomes an important quality characteristic. Some mobile applications are safety-critical and require extremely high reliability. Reliability equates to the robustness of the software including how well it handles faults (fault tolerance), how quickly it can recover from a problem if one should occur, and how consistent it is in providing the same result for the same actions.

3.3.4.1 Reliability Testing for Mobile Software

Testing for reliability requires causing failures and verifying that the software correctly detects the failure and either handles it or recovers gracefully. It is important that mobile software is able to reconnect when connections are lost and continue processing without losing any transactional data. Mobile devices, by definition, move around. They go to places with poor network connectivity. They go in tunnels and underground. They go in airplanes. At a minimum, mobile devices go everywhere people go and this creates a large set of potential reliability issues.

While mobile device reliability testing must be concerned with such things as temperature tolerance, impact, submersion, extreme heat and cold, and so forth, the applications running on the device must be able to respond to the effects of these conditions on the device including device failure.

3.3.4.2 Reliability Testing Approaches

Mobile applications must be tested for the same reliability issues as any other application. This includes insufficient memory or other resource constraints, hardware failure, and network or communications failures.

In addition to the traditional reliability tests, testing must also include the ability of the application to handle low battery levels, shutdown conditions, complete power failures and similar power related issues. Because mobile devices usually run on battery power, power failure poses a risk with a higher likelihood than would be expected for a traditional application. Similarly, issues with network connection, disconnection and reconnection (including switching such as from cellular to WiFi) must all be tested because of the high likelihood of occurrence. Mobile devices tend to be shut down less frequently than traditional computers which results in a longer period of operation which can allow problems such as memory leaks to become more apparent.

Reliability can also be measured by determining how long a mobile application can operate continuously without failure (or without recharging the battery). This can be performed by creating and performing simple automated tests and measuring the mean time between failures (MTBF). When failures occur, they can be captured in reliability failure scenarios. These scenarios can be provided to the development team to help them design stronger mobile applications that will prevent or handle the defined failures. It will also allow the developers to create recovery procedures if the failures do occur.

4 Environments and Tools - 285 mins.

Keywords

emulator, native device, simulator

Learning Objectives for Environments and Tools

4.1 Tools

MOB-4.1.1 (K1) Recall the expected capabilities for mobile application testing tools
MOB-4.1.2 (K2) Explain the use of generic tools in testing mobile applications

4.2 Environments and Protocols

MOB-4.2.1 (K1) Recall the sources of data for a mobile application

4.3 Specific Application-Based Environment Considerations

MOB-4.3.1 (K2) Explain the differences between browser-based and native device applications

4.4 Real Devices, Simulators, Emulators and the Cloud

MOB-4.4.1 (K2) Explain why testing is not conducted entirely on real devices
MOB-4.4.2 (K3) For a given mobile testing project, determine how and when to use simulators/emulators during testing
MOB-4.4.3 (K1) Recall how to verify the reliability of a simulator/emulator
MOB-4.4.4 (K3) For a given mobile testing project, determine how and when to use cloud-based testing

4.5 Performance Test Tools and Support

MOB-4.5.1 (K2) Explain how the cloud can be used to support performance testing
MOB-4.5.2 (K2) Explain the types of data a performance tool needs to be able to create and track

Common Learning Objectives

The following learning objective relates to content covered in more than one section of this chapter.

MOB-4.x.1 (K3) For a given mobile testing project, select the appropriate tools and environments for testing

4.1 Tools

The mobile device and application market is expanding rapidly. Fortunately, the tools for testing the mobile applications are keeping up with the explosion. The tester now is confronted with trying to select the best tool from a large set of changing tools with varying capabilities and reliability. This section will better prepare the tester for understanding the tool options and provide information to help select the best tool for the specific application and environment.

4.1.1 Application to Mobile

There are plenty of test tools on the market, but it's important to discern between tools that provide general testing support and those that are specifically aligned to testing mobile applications. In general, tools that are focused on mobile testing should be able to do the following:

- Adapt to different environments and protocols
- Simulate a native device
- Support testing across iOS, Android and other operating systems
- Simulate multiple users simultaneously
- Support mixed and varying locations of devices
- Support or simulate networks of varying speeds and quality
- Simulate or provide connections that can be disconnected and reconnected

When seeking a tool, the tester must understand the capabilities of the application, the environments to be supported, and the testing requirements in order to select the tool most suited to their needs. Because the market is changing so quickly it is important to use good tool selection processes and ensure the vendor has created and will support a quality product. Pilot tests are needed to ensure the tool will work in the specific environment. For a more complete discussion on tool selection and deployment, see [ISTQB_FL_SYL]. Even if the purchase price of a tool is low (or free) the organization will still invest a considerable amount of money to implement and use the tool. Proper evaluation processes need to be followed for any tool procurement, even a low cost one.

Tools should be evaluated for ease of use. As the mobile testing tools increase on the market, the usability will become better. Sometimes it makes sense to wait for a later version of a tool if it will have a vastly improved interface. As with any tool, remember the skill sets of the tool users and ensure the tool will provide the necessary functionality in a way that is accessible to the tester.

4.1.2 Generic Tools

Generic tools are still useful when testing mobile applications. For example, test management tools, defect management tools, and requirements management tools are all still needed. Build tools, continuous integration/deployment and unit testing tools are still needed to support the development process. Since many mobile applications also have a backend component, tools used in the testing of the software running on application servers, web servers, and database servers are still needed. In general, the background or supporting software will be tested in the same way for mobile applications as it would be for client or web applications.

4.1.3 Commercial or Open Source Tools

Commercial tools are those made by a company, for profit. While these may be considered by some to be more reliable, they tend to lag behind the market needs. Open source tools are created by interested individuals or communities who have created a tool for a specific need. Open source tools tend to be focused on solving a particular problem whereas commercial tools are designed to address a wide range of capabilities. In the mobile application testing world, open source tools are readily available with a varying focus and capability set. Commercial tools are also available but may lag behind a bit in adding coverage for new capabilities and technologies. Before selecting either type of tool, due diligence is required to ensure it is the most appropriate tool for the job. It is important to consider tool support, maintenance, applicability and ability to grow with the industry as well as cost and usability.

4.2 Environments and Protocols

4.2.1 Environment Considerations

A mobile application is expected to work in numerous environments. Each of those environments can have its own particular features. At a minimum, the following areas need to be considered when test environments are being selected and established.

4.2.1.1 Connectivity

Devices are able to connect via WiFi, cellular networks and may talk to each other via Bluetooth technology and various other means. When testing the connectivity, it is important to test for disruptions in the connection, reconnection capabilities, the ability of the application to continue when data loss has occurred during the transmission, and lost connections. It must also be verified that the device can switch between connections without losing data or impacting the user. For example, a device may switch from WiFi to cellular, or may switch from a slower 3G network to a faster one. Because devices are mobile, they should be selecting the best connection available. Testing the various connectivity options requires significant network resources and configuration capabilities.

4.2.1.2 Memory

Device memory varies widely. Tablets, smart phones and other devices come with a range of memory options. New devices generally start on the market with a lower memory capacity and increase the capacity as newer models are introduced. Devices on the market may also be able to add more memory. It is usually a safe assumption that if an application fits within the memory on an existing device, it will still fit on future versions of that device that have greater capacity. New devices of course support new features and peripherals which may be competing for memory usage, but that is no different from the memory competition that will always occur.

Testing for efficient memory usage (efficiency testing) and ability to handle low memory situations (fault tolerance) is important for mobile applications running in a shared environment with competing processes. Memory management must be monitored to ensure no memory leakage is occurring due to allocated memory not being released and that no memory corruption is occurring.

4.2.1.3 Performance

The performance of the test environment is an important consideration for creating valid test results. The performance of the test environment should mimic the performance of the production environment, including communication interruptions, reconnections and network traffic. Because communication is at the core of the performance for mobile applications, the test environment must provide a realistic communication interface including an ability to introduce and control the problems that are likely to be encountered in production such as weak connections, timeout errors, and so forth.

4.2.1.4 Device Capabilities and Features

Devices vary considerably in their capabilities and features. While application testing should not include testing all the capabilities of the device, it is important to understand how the features of the device may interact or affect the application. For example, if the application requires the use of an accelerometer and gyroscope to determine the orientation of the device to display the application interface correctly, testing must include devices with various versions of these features any of which may have different interfaces, as well as devices without the features or with malfunctioning features. The more features of the device used by the application, the larger the testing pool of devices becomes.

Features and capabilities to be considered when determining the proper test environment for the application include:

- Screen size for display
- Screen lighting
- Geolocation
- Telephony

- Accelerometer
- Gyroscope
- Magnetometer

In addition, the capability of a device to install the application over-the-air (OTA) must be considered when determining how to download releases and updates to the device. The OTA capability may limit the size of the downloads and the installation routines must have strong recovery for interruptions, partial downloads and missed updates. An update must not adversely affect the functioning of the device or result in data loss.

As was mentioned in the techniques section, combinatorial testing techniques such as pairwise testing can help reduce the potential number of test environments by determining representative combinations of capabilities and features that do not interact. Decision tables can be used to determine necessary combinations of capabilities and features that do interact.

4.2.1.5 Data Handling

Another environmental consideration is the data that will be used by the mobile application. In general, data used by a mobile application can come from one of four sources:

- The backend system (e.g., database servers)
- The device itself (e.g., GPS location)
- The user (via some type of input e.g., text, selections, UI interaction)
- Another connected device (e.g., a PC connected via USB)

Testing for mobile applications must consider receiving data, sending data, and storing data on the device. Data must be handled with the appropriate level of security and reliability. When data is stored on the device, special care must be made to ensure the data is safe and protected from unauthorized use.

4.2.1.6 Device Location

When testing mobile applications, the tester has to consider access to the physical or simulated device. Since mobile application testing often requires testing across a number of devices, access to these devices must be determined during the test planning phase. There are a number of ways to access multiple devices and in some cases, multiple testers on multiple devices. The following list includes some of the more common approaches to procuring a large number of devices for testing:

- Actual physical devices co-located with the testers
- Open Device Labs [OpenDeviceLab]
- Crowdsourcing (e.g., utest.com)
- Remote device labs (e.g., from the vendor or an external organization)
- Virtual environments (e.g., simulators, emulators and cloud-based)

Acquiring a large set of devices and keeping that set current with new models and versions can be cost-prohibitive. As a result, using a remote test environment furnished by tool vendors or virtual environments is often a more feasible approach.

4.2.2 Protocols

Different devices may use different communication protocols. Testing tools, particularly load testing tools, may not support all protocols. The tester must understand what protocol is used by the devices to be tested to ensure there are no incompatibilities with the tools that will be used during testing. If simulators will be used, the simulator must be able to simulate the use of the device's protocol.

4.3 Specific Application-Based Environment Considerations

4.3.1 Browser-based Applications

Browser-based mobile applications are designed to run on a set of supported browsers that run on the device. Some of these may give the user the option to switch to the mobile version of the application or stay with the standard web application. One advantage to creating browser-based applications is the built in portability. Any device that can run a browser should be able to run the application. This should, in theory, limit the testing requirements to testing with the various supported browsers on any type of device. There are, however, specific considerations for testing when a browser-based application is also considered to be a mobile application. For example, different browsers and versions vary widely in how they handle certain components, java script, and cascading style sheets (CSS). Some plug-ins may not be available on a mobile device (e.g., Flash).

4.3.1.1 Considerations for Usability and Performance

Because these applications may not have been optimized for a mobile device, there may be issues with usability and performance. In particular, font sizes, navigation and screen layout may not be user-friendly when viewed on a smart phone. While many of these applications have an option to switch to the “mobile” version, that option may not be apparent to the user who is confronted with tiny fonts and only a portion of the main window.

Performance may also be an issue for an application that was not designed for the mobile environment. As was discussed in the performance testing section, the tester will still need to test for performance but should also watch for connectivity issues that might impact reliability and performance.

4.3.1.2 Browser Version Support

As with any web application, the application must be tested with the supported browser versions. When the application is also intended to be deployed in the mobile world, additional browser versions may be required and, more importantly, the frequency of some browsers may differ from the traditional platforms. For example, the PC environment may see a dominance of one type of browser where mobile devices may see a different one. When this happens, testing prioritization must shift proportionally to the more frequently used browsers.

4.3.2 Native Device Applications

An application that is built for a native device is generally using specific features and capabilities of that device and its operating system to deliver functionality. When this happens, the same application generally is not portable to another environment. There are a number of reasons for developing native device applications. In addition to being able to take full advantage of the device capabilities, the developer is also able to tune the user interface to a more specific market. This approach is also used for devices that do not have the capability to run a browser or those that must work without Internet connectivity.

4.3.2.1 Good Simulator or the Real Device

With native device applications, the simulator must be designed specifically for that device. If simulators are not available, then the testing must occur on the real device. This, of course, may become costly if the device is difficult to obtain or performance testing is required for a large number of the devices. The tester should work with the developer to understand what aspects of the application require the specific device and what could possibly be tested with a generic simulator.

4.3.2.2 Tool Support

Depending on the market popularity of a native device, tool support may not be available, or may not be available soon enough for the testing of new applications. Tool support will follow the market. In general, tools will be developed first for the major market players such as iOS, Android and Windows.

New devices may not have the tool support needed to procure simulators, conduct performance tests or to support test automation.

4.3.3 Hybrid Applications

A hybrid application may use specific features and capabilities of a device via plug-ins. Because device compatibility depends on the framework used to develop the application, testing across different devices must still be done as the framework may not be defect-free. Generally, all the testing needed for native applications and for browser-based applications is needed for a hybrid application.

4.4 Real Devices, Simulators, Emulators and the Cloud

Testers, working with developers, must determine the most appropriate test platform. This can be real devices, simulators of devices, emulators, or a mix of the three.

4.4.1 Real Devices

By definition, real devices are the most realistic, providing the most accurate test environment. Real devices will provide the production environment and will allow the tester to observe any device specific issues that might be missed with a simulator. Devices have hundreds of differences between each other. They also have their own defects [Firtman], such as inconsistent sensitivity to user input (e.g., not detecting pressure at the edge of the screen) or physical issues (e.g., buttons that are not reliable or consistent). Simulators will always lag behind the real devices because it takes time to build and test quality simulators.

Real devices, although the best test platforms, may be difficult to obtain. It may also be difficult to create test automation and performance testing that will work with real devices. Generally, it is best to test with a mix of platforms, using the real devices as a sample set for usability, performance sampling, and general functional tests. Real devices are often used for comparison against the simulated devices to ensure the simulators are giving “real” responses.

Usability testing should always be conducted on the real device in order to give a proper and full assessment of the user experience. Device differences can affect usability and even though the application is the same, it may have a different usability level on different devices.

Depending on the application and the target platform, it may be possible to test with devices that are similar in capability to the target. This is sometimes done when new operating system versions are available or new features are available on new devices, but are not leveraged by the application.

4.4.2 Simulators

A simulator is a program that simulates some aspects of a device. It does not emulate the hardware itself and may not simulate all the device responses and activities. It does not work on the same operating system as the device.

Simulators are sometimes supplied by device manufacturers to help developers test applications. Since a device will be more popular if it has many applications, it is in the best interest of the manufacturer to supply a simulator, and a good one. However, simulators are not necessarily good or reliable and may not be a good representation of the real device.

4.4.2.1 Buy or Build

If a reliable simulator is not available, the development organization may choose to write their own. While this will enable them to create exactly what they need to develop and test the application, the simulator itself must be tested. Building a simulator is a development project in itself and requires

analysis, design, architecture, development, testing and, of course, documentation. Also a simulator must stay current with changes made to the real device or testing with the simulator becomes invalid.

4.4.2.2 Verify Simulator Reliability

Before relying on the simulator, it is important to ensure that the simulator is giving correct responses to inputs. This is done by providing the same inputs to the real device and the simulator and verifying if the results are the same. If not, the simulator is not reliable. This is true for functionality as well as performance. A slow simulator that is functionally correct (or a fast one) can still be used for functional testing, but further testing will be needed on a real device to ensure that the response time differences do not affect the application.

4.4.2.3 Using Simulators for Performance and Load Testing

Simulators are commonly used for load generation and performance evaluation. Because simulators are software rather than hardware, large numbers can be created and run without additional costs of procurement. As with any performance testing, the tester should be sure the activities per simulator are equal to the expected activities per real device in order to create accurate load and performance reports.

4.4.3 Emulators

Emulators are used to provide the functionality of the device itself including software, hardware and operating systems. This is necessary for certain applications that may use various device components such as cameras or special screen controls. Emulators are usually written by device manufacturers although some are available from other sources. It is difficult to test an emulator for proper functioning without knowing the internals of the real device. If an emulator is to be used, the tester should ensure that it is from a reliable source and that it has been thoroughly tested. Spot checking responses against real device responses is a good idea and will help verify that the version of the emulator being used corresponds to the target device.

4.4.4 Cloud

There are a number of cloud alternatives for mobile application testing. These include the following:

- Cloud hosted appliances – Appliances or devices exist in the cloud and can be accessed via manual or automated test. This allows access to many different types of devices. These devices can be used for functional testing as well as performance and usability testing.
- Cloud hosted agents – Software can run in the cloud that simulates users from all over the world. This allows a site to verify what happens to its backend when many users of mobile devices from many types of networks, use the application. This is sometimes done with device simulators in the cloud and sometimes done with real devices in the cloud.
- Cloud network simulators – When doing testing in the cloud, network simulators can be used to simulate various network configurations, speeds and error conditions. This allows a realistic test environment to be created with varying network types.
- Cloud protocol simulators – Since devices may communicate via different protocols, the protocol simulators are used to simulate those protocols. This allows an organization to test their application with varying protocols or to do performance testing across multiple protocols.

Any cloud solution for testing must consider the reliability of the cloud environment, the realism of the environment (which often is determined by the configurability) and the accessibility. In addition, there are some security concerns with using cloud environments for testing, particularly for new and innovative applications and devices

4.5 Performance Test Tools and Support

Just as the mobile devices have exploded into the market, so have the tools to support testing them, particularly for performance. Cloud solutions allow access to a large number of devices, networks and protocols, allowing load to be developed on a system from a variety of cloud-based devices or device

simulators. A good tool is not enough to do effective performance testing though. Performance testing must be targeted to the operational profiles that matter and the loads that are expected to be experienced by the system, as discussed in Section 3.3.1.

When selecting a performance testing tool, the tester should consider the tool's ability to create and track the following:

- Information flowing between the device and the servers, such as
  - Transaction data
  - GPS information
  - Images
- Volume and frequency
  - Connection/disconnection patterns
  - Activity bursts
- Usage patterns
  - Variances for time of day/week
  - Uses for business devices versus personal devices
  - Peak usage times (e.g., daily, seasonal)

The ability of the tool to control and monitor this information is critical to the success of the performance testing effort. Testing with the wrong tool can result in wasted time or, worse, incorrect results.

There are many good commercial tools available for performance testing. Some of the traditional tools are adding support for mobile devices. New, targeted mobile performance testing tools are entering the market frequently. When making a tool decision, it is important to remember to assess the capabilities needed today and tomorrow. It is a good practice to plan for success and assume the application's user base will grow dramatically, much faster than with traditional software.

4.6 Test Automation

Test automation is not optional in mobile application testing; it is a requirement. As more tools are joining the market, test automation is feasible and maintainable only if well designed and planned. Mobile application test automation projects must be sure to consider the following:

- Base the tests on realistic usage patterns
- Understand and test the interactions between the user and the device
- Understand and test the interactions between the device and the servers
- Isolate the data from the test automation script by using data-driven or keyword-driven approaches [ISTQB_FL_SYL]
- Ability to control the real device
- Develop for maintainability
- Version control the test cases so that older versions of the test automation are available to check a maintenance release

Test automation is vital for mobile application testing and requires good planning, solid design and careful implementation. Maintainability must be designed into the test automation code and good practices such as isolating the data will help ensure maintainability. Because mobile applications change so rapidly, maintainability is an even higher priority in mobile application test automation than it is for traditional applications. Test automation can be as valuable as the software it is testing. Version control, good coding practices, and quality requirements apply to the test automation code. Ideally, test automation is built as the application is built, enabling automated testing of the continuously integrated and deployed application.

Test automation with real devices is difficult. Test automation with simulators (or emulators) is the more common approach and requires that simulators be built to support automation by providing the proper interfaces to allow the automation to communicate with the simulator. Since a simulator may not have buttons to push or a screen to touch, it needs to provide a capability for the test automation to send commands that would accomplish the same result as a user interacting with a real device. There are tools on the market that drive automated tests to actual devices, but the cost may be a factor for smaller development organizations and using simulators may be the more practical solution.

4.6.1 Tool Support

As with performance testing tools, test automation tools for mobile applications are also rapidly entering the market. Careful tool evaluation is required, but it makes sense to look to the new tools with more finely tuned abilities for mobile applications rather than some of the traditional test automation tools that adapt more slowly to the new markets. Because mobile devices and their applications will continue to evolve rapidly, any test automation tools must be able to adapt as well. Test automation is a significant investment and buying the wrong tool can be a costly mistake.

4.6.1.1 Pick the Environment

Tightly coupled with the proper tool selection is the proper environment selection. Test automation must be targeted to an environment. That environment may include:

- Real devices
- Simulated devices
- Cloud hosted devices or user agents
- Combination of any of the above

Focusing the test automation on the target environment will help the tool selection process and will help ensure that the tool will be able to support the testing as the testing environments expand. Even if a cloud solution is not appropriate for the organization today, it may become necessary at a later date. Since test automation should be designed to last for several years, those types of environment changes must be considered early.

4.6.1.2 Support for Coordination

It may be necessary for a tool to be able to support transactions being sent to and received from multiple devices and simulators. The tool may need the ability to correlate this information, particularly for server tests, to understand what is happening on the system at the time the test automation is being executed. This coordination may include:

- Number of transactions
- Timing of transactions
- Types of transactions
- Summarized reporting

Tools that are not capable of managing this coordination will cause the test team to spend considerable manual effort to set up tests and analyze results.

4.6.2 Skills Needed

As with any test automation project, programming and scripting skills are needed to develop high quality, maintainable test scripts. It can be misleading to think that a mobile application will have a short life in production before being upgraded and therefore the test automation code will only have a short life as well. Good test automation can grow with the product and can provide good regression testing for the old features when new features are introduced. In order for this to happen, the test automation architecture must be robust – built to last and grow.

In the mobile application world, the capability sets will continue to grow. Automation will not be stable – it will always be expanding to accommodate new features of the device and the applications. Tools may lag behind device development. It is a reasonable expectation that programming or scripting will

be required to bridge the gaps between tool capabilities and device capabilities. Before coding to fill a gap, the tester should check if a reliable tool is available. Tools are developed and deployed very quickly, so frequent investigation is warranted to avoid unnecessary effort.

5 Future-Proofing – 135 mins.

Keywords

none

Learning Objectives for Future-Proofing

5.1 Expect Rapid Growth

MOB-5.1.1 (K1) Recall ways in which the mobile application and device market will expand
MOB-5.1.2 (K1) Recall areas in which user expectations will increase

5.2 Build for Change

MOB-5.2.1 (K2) Summarize the considerations for building a flexible testing framework
MOB-5.2.2 (K4) Analyze a given mobile testing project and determine the appropriate activities to reduce maintenance costs while enabling wide product adoption

5.3 Plan for the Future

MOB-5.3.1 (K2) Explain how lifecycle models are likely to change and how this will affect testing

5.4 Anticipating the Future

MOB-5.4.1 (K1) Recall the ways in which testers will need to adapt

5.1 Expect Rapid Growth

If the past is any indication, the mobile application and device market will continue to expand. This means there will be more types of devices and variations of existing devices. The devices will have more capabilities. There will be more applications and those applications will continue to become more feature rich to take advantage of device capabilities and increased memory.

But that's not the only areas for growth. The number of users, the variety of users and the expected usage of these devices will also grow. As the devices become more and more a part of personal and business life, user expectations will continue to increase in the following areas:

- High reliability
- Excellent usability
- High performance
- Consistent experience
- Portability
- Fast turnaround for fixes and new features

As a result of these ever-increasing expectations, software testers will need to become more adept at defining and executing the necessary tests while also facilitating a rapid time to market. This will require leveraging the right tools, picking the appropriate environments and using new approaches to deal with the large number of devices and user types.

5.2 Build for Change

As profit margins decrease due to competition in the market, pressure will be put on the development and testing teams to produce high quality, maintainable products quickly. Testing will need to engage

early with development to plan out the testing, procure the tools and environments, and upskill as needed.

5.2.1 Architect the Testing

It will not be enough for the developers to plan for change. The test approach must also be architected for change. Some of the considerations for this flexible framework include:

- Implementing or utilizing test environments that can be quickly assembled and disassembled
- Utilizing the most appropriate tools for the tasks
- Implementing maintainable test automation (e.g., keyword-driven)
- Designing the load testing approach at the beginning of a project
- Conducting load testing throughout the software development lifecycle
- Maintaining a reasonable set of representative devices
- Accommodating a risk-based testing approach
- Providing a framework to support crowd-sourcing
- Providing a strong ROI that supports the shorter lifecycle

Tools must be selected for flexibility and ability to adapt to the changing market. Tool vendors must be highly responsive to the market changes. Relying on past reputation will not be sufficient in a market that is moving quickly and has many tool options.

While re-usability is always a goal, in the short lifecycles of the mobile applications, achieving time to market with a quality product may supersede the value of having a fully reusable testing solution. Re-usability will likely save money in the long run, but too much time spent creating a fully re-usable testing solution may result in an unacceptable delay in product release. Investments must be justified by the expected lifetime of the product. Today's best smart phone is likely to be tomorrow's paperweight, so the investment in the testing must be justified.

5.2.2 Enable Efficient Maintenance

While maintainability may not be the ultimate goal, efficient maintainability is a requirement. The test environments, tools, and testware must be able to be efficiently maintained or replaced. It may be that an environment is needed for an initial release of an application but will not be needed for the next release. This means the investment in the environment must be justified by the risk mitigation that is expected to be achieved from testing in that environment. Unlike traditional software where longer term planning is justified, mobile application software may have a much shorter lifecycle so the maintenance of the test environment and the need for reuse may also be similarly limited.

5.2.3 Select Tools for Flexibility

If tool vendors cannot be responsive to the market, purchasing their tools and creating significant test assets with those tools would be foolish. The tools must be as flexible as the products will be and any investment in test automation or even test management must conform to the expected lifecycle of the product. Inexpensive tools that can be used to create less resilient test automation may be justified over heavier weight tools that will create a product that will provide long term maintainability, particularly if the product is not planned to be used long term.

When designing testware, it may make sense to design it to work with a simulator rather than a particular device. This may offer more flexibility in the long run as devices evolve and add more features. Developing to a simulator will also allow test execution prior to the availability of the real device.

5.2.4 Select Partners Carefully

When third party relationships are formed for testing, these relationships should be built on the assumption that there will be many releases. Long term relationships will allow more flexibility in

meeting market needs without incurring downtime due to contract negotiations, requirements debates and so forth. If a third party relationship will be formed, it is important to ensure that the partner will be able to keep up with the changes in the industry. Expect a demonstration of flexibility and ability to adapt to changing markets. A partner that is locked into a particular tool set or a particular methodology may not provide the agility needed to cope with the rapidly changing market.

These relationships may include outsource testing, device labs, cloud environment vendors and so forth. Because building a full mobile application test capability may be cost prohibitive, partnerships may be the most viable way to meet variable needs.

5.3 Plan for the Future

5.3.1 Lifecycle Models

It is likely that new lifecycle models may be introduced as a result of the mobile application and device requirements for testing. Agile and iterative lifecycles already dominate the industry, but new leaner methodologies may become popular. The tester must remember that any lifecycle model will require some adaptation and the tester must understand the moment of involvement and level of involvement that will be expected in the various models.

Timescales for the development and testing of applications have never been so short. The market demand has never been so high. Of course, the opportunities are plentiful. But, an organization's reputation can be badly damaged by poor quality products and ill-considered feature sets. The tester must expect to work with the developers to ensure the best possible product is being released within the given timeframe.

5.3.2 Alternative Testing

While new lifecycle models may be on the horizon, there will still be a need for efficient testing. This means taking a lightweight approach to deliver a proper ROI. Automation and performance testing will be required and the tools must be appropriate to the needs, schedules and budgets. Security will always be a consideration and security testing tools will likely adapt to fill the mobile market as well.

Testing in the cloud is likely to become a commonplace practice where device simulators and user simulators are used to create and test a realistic variety of transactions and system load. Crowd-sourcing, testing-in-the-wild, and other forms of outsourcing are likely to continue to grow and will continue to present test management challenges.

As the tester community expands to non-testers, automated error reporting and screen capture will become more commonplace to help developers track and diagnose issues that occur in production. This information should also be fed back through the testing process to ensure improvements are made to close any testing gaps.

5.4 Anticipating the Future

Planning testing practices, processes, and tools for use two to three years in advance is difficult in the mobile application space. New applications and devices emerge daily and competition will continue to drive the market and lifecycle.

Testers need to be ready and willing to adopt new technologies, investigate new tools and learn more efficient and leaner testing methodologies. Some products and practices will be unsuccessful but good research should allow testers to select the right approaches and the best tools.

6 References

6.1 ISTQB Documents

- [ISTQB_ATA_SYL] ISTQB Advanced Test Analyst Syllabus
- [ISTQB_FL_OVIEW] ISTQB Foundation Level Overview
- [ISTQB_FL_SYL] ISTQB Foundation Level Syllabus
- [ISTQB_GLOSSARY] Standard glossary of terms used in Software Testing

6.2 Trademarks

The following registered trademarks and service marks are used in this document:

- ASTQB® is a registered trademark of the American Software Testing Qualifications Board
- ISTQB® is a registered trademark of the International Software Testing Qualifications Board
- PRISMA® is a registered trademark of Erik van Veenendahl
- Teststorming™ is a trademark of Rice Consulting Services, Inc. and is used by permission, with permission granted to freely use Teststorming in the creation of courses and other derivative works with attribution to Rice Consulting Services, Inc.

6.3 Books

[Black09]: Rex Black, “Managing the Testing Process”, John Wiley and Sons, 2009, ISBN 978-0-470-40415-7

[Firtman]: Maximiliano Firtman, “Programming the Mobile Web”, O'Reilly Media; Second Edition edition (April 8, 2013), ISBN-10: 1449334970

[vanVeenendaal12]: Erik van Veenendaal, “Practical risk-based testing – The PRISMA approach”, UTN Publishers, The Netherlands, ISBN 9789490986070

[Whittaker]: James Whittaker, Jason Arbon, Jeff Carollo, “How Google Tests Software”, Addison-Wesley Professional; 1 edition (April 2, 2012), ISBN-10: 0321803027

6.4 Other References

The following references point to information available on the Internet. Even though these references were checked at the time of publication of this syllabus, the ASTQB cannot be held responsible if the references are not available anymore. The ASTQB is not endorsing any of these sites or their products. The references are provided as a source of information only.

[InfoQ] http://www.infoq.com/news/2014/10/world-quality-report

[Paskal] www.gregpaskal.com

[OpenDeviceLab] www.opendevicelab.com

[OWASP] https://www.owasp.org/index.php/Projects/OWASP_Mobile_Security_Project_-_Top_Ten_Mobile_Risks

[Nielsen] Jakob Nielsen, “Why you only need to test with 5 testers”, www.useit.com

[Rice] “Teststorming”. http://www.riceconsulting.com/home/index.php/General-Testing-Articles/teststorming-a-collaborative-approach-to-software-test-design.html

[Samsung] https://developer.samsung.com/remotetestlab

[Webtrends] http://www.webtrends.com/blog/2010/04/top-metrics-for-mobile-apps-measure-what-matters/