# The Essentials of Mobile App Testing

An AQuA Guide.  V1 April 2013

App Quality Alliance

We're often asked to explain the differences between types of testing.

In this document we look at how the testing of mobile applications helps to achieve quality.

We explore a typical way that an app is developed, look at the testing stages involved, answer some of the frequent questions concerning testing, and provide a definition of the common testing terms.

In conclusion we highlight what AQuA can offer in assistance.

The document has been compiled by AQuA, with additional input provided by the AQuA-approved Test Houses at April 2013.  We'll be revising it continually, so if you have any input for the next version, please do let us know.  Thank you.

We hope you find it useful.

## What is 'quality', and who is interested?

Quality is about developing a product that fits the customer need and works well in the environment, given the constraints within which it exists.

Or is it?

Many definitions of quality exist but most of them apply to production line and factory processes.  In that environment, quality is all about keeping a defined production process running, and finding an acceptable level of product faults.

The Chartered Quality Institute in the UK lists 20 different definitions of quality, but settles on the following as its main banner heading:

**What is quality?**

Quality management is an organisation-wide approach to understanding precisely what customers need and consistently delivering accurate solutions within budget, on time and with the minimum loss to society

*© 2013 the CQI*

There are usually four groups of people who are interested, and they tend to see different aspects as the most important:

### The Developer

Wants to achieve good ratings in the eyes of the user or satisfaction in the eyes of the commissioner of the app.  Most developers are good at creative  development, but few have a background or experience in testing and Quality Assurance.

### The Commissioning Party:

Wants to see that the developer has done a good job and that the brand for their product or service is well-represented by the mobile app.

**The Distribution Channel:** (Shop, Network Operator, device manufacturers)
Shops and network operators often receive the backlash of a poor quality app, either through returned handsets, complaints to customer service or even congestion on the data network. Improved quality of applications is better for all parties involved in providing the infrastructure.

**The End User**
Wants to have an experience as advertised by the app. They don't have time for crashing apps, or poorly designed user experiences. There are hundreds of thousands of apps out there, so they really need some help to find the good ones.

So, although the definition of 'quality' may change, and the reasons for needing it differ depending on who you're asking, the importance of quality isn't in dispute.

## To test or not to test ...

Assuming that mobile app developers are trying to achieve high quality apps, how can the quality be measured or proved?

As with many things, it is hard to positively prove quality. It is however possible to demonstrate lack of quality by testing an app and showing that it fails.

If you can't prove quality, but you can find failures by testing, the question then is how much time and effort should you spend to test an app? Clearly there must be an appropriate level of testing, neither too much nor too little. The choice of this level is down to managing the risk of not finding a fault, the risk of something going wrong when the app is in the user's hands and the subsequent damage this may cause to your brand or reputation.

In a fast-moving and crowded market such as we have today, mobile apps need to be developed and deployed rapidly and effectively at optimal cost. Whilst there is no time for some of the more structured and formal methodologies that may be employed in, for example, safety critical software, the scope and scale of the mobile app market and the speed of comment and criticism of a poor app, means that no-one can afford to ignore the quality of the app and some structured process is necessary for success.

The big question is how to balance the work required for testing against the risks of not testing. How much to test and how and when?

## Typical development and testing cycle for a mobile app

Let's now look at the typical development cycle for a mobile app. This shows where testing needs to be done, and how the developer can gauge how much testing of each type to do.

(This section uses terms for different aspects of testing. A full list and comparison of these different terms can be found later in the document.)

## In the Developer's "Lab": writing the code

**After the functional requirements for the app are decided, the developer will start to design the app and then generate the code and graphic images that are needed.**

To help with the design, there are Best Practice guidelines available from AT&T, AQuA and GSMA and others, looking at aspects such as common errors, memory usage, network efficiency and battery optimisation.

Typically for Android, the code will be produced in Java using an Integrated Development Environment (IDE) - typically Eclipse. The Android SDK adds the necessary definitions for the APIs in the platform. Clearly the version of the SDK that is used will determine the compatibility of the generated app with the various versions of the Android OS.

The App will be written as a series of components, each one of which can be tested against the designed functionality by using a tool like Junit or Xcode that allows the developer to drive the component and see the output. This **Unit Testing** usually focuses on functional testing, proving that the component functions as designed.

Once the app is taking shape, the developer may choose to load it onto a device that is locally connected in Debug Mode to the IDE and do some on-device debugging.
In this phase the developer has progressed from solely unit testing of the components to add in some **Integration Testing**.

The process continues as the developer builds the various functions of the app and locally tests that the functionality works as intended.
The developer is now adding some end-to-end **System Testing**, usually focused on testing the key functionality and interoperability with any external components that may be used.
There is a range of tools that can be used to help with this functional System Testing. Typically the tools will allow the developer to repeat functional tests and automation of this can be very beneficial.

(Experienced developers will sometimes add a level of complexity into the above process. Sometimes different versions of the app targeted at different handsets may be developed at the same time as the core functional development. This is usually not advised for inexperienced developers as it usually results in a lot of extra testing. We will continue to presume on developing the app for a single device first and later testing for compatibility on other devices.)

At some point the developer will decide that they are happy with the App, that it works to their satisfaction and that they are ready to let the app out of the lab.
The developer could be tempted to put it straight into the distribution channel, or give it for approval to the person who commissioned them to write it. This would be quite a large risk as there is a whole range of aspects not yet proven as we shall see…

Coding complete.

## In the "Test Lab": Getting the App ready for market

**This next stage of development is one that is often left out as many developers would regard it as unnecessary since they have already proven the functionality to themselves**.

This is a **Quality Assurance** (QA) testing stage; a series of tests designed to go beyond the purely functional tests and preferably done by a separate 'test team'.

The reason for a separate test team is that the developer themself is usually too close to the application. If you know the right sequence of events, or the right keys to press, then you will never find out what happens when a real user has the application and follows their own view of the appropriate sequence.  A separate specialist tester or team of testers are usually skilled in finding such issues.

In many developers' minds, QA testing appears as a bottomless pit of pain, where many unnecessary tests are carried out for no good or clear reason.
In truth, a well-managed QA testing process is invaluable and will catch issues that would otherwise spoil the reputation of a good app and the developer, disappointing the users who would then give low ratings and turn to the opposition's offers instead.

Once the app is ready for QA testing, the AQuA Testing Criteria can be employed here to achieve a good level of QA that covers the critical aspects of each of the testing objectives.

Using the AQuA Testing Criteria to form a QA test plan will catch the majority of common errors.  The test cases in the AQuA Testing Criteria have been developed over time with input from experienced Operators, Handset manufacturers, Test Houses and developers.  They are designed to be 'Right Sized' to have enough test cases to prevent errors, but not so much that they are over-bearing, and define the minimum level of testing that should be done to make sure an app will work well.

AQuA specifies that the tests should be carried out on a real device.  Clearly this device could be remote (see below for functional system test tools / remote device tools), but that has drawbacks for tests that require physical intervention (such as removal and re-insertion of the memory card).

AQuA classifies different types of apps by functional complexity, which is defined by the technical permissions requested.  A 'simple' app for example typically might be one that is stand-alone app, or just use simple web protocols to retrieve data.  Subsets of the AQuA Testing Criteria have been defined for each such type of app.

Whilst running a complex app against most of the test cases within the Criteria should take approximately half a day to do on one device.  A simple app can usually be tested using the appropriate subset of test cases in less than a couple of hours.  Clearly if the testing finds lots of errors it will take longer!

## In the Customer's hand: Customer Acceptance Testing

**If the customer has commissioned the app, they may have their own structured set of tests that they will carry out before accepting the app and paying for it.**

If the customer is an individual, having bought the app from a store, they may do no testing, or they may quickly try out the app, which, although the customer is unaware of this, is a form of acceptance testing.  If the app doesn't impress in that first few minutes, they will give up, ask for a refund (which they may get if they ask within 15 minutes of downloading the app from certain app stores) and may provide a bad review…

## Additional testing?

The AQUA Testing Criteria defines a minimum level of testing to make sure that the app will work well, but what other additional testing might be worthwhile to add on top of that?  Testing is about risk management.  It's carried out to reduce the risk of failure, and so the amount of additional testing you can do before releasing your app to your customers depends on the risks involved, and the cost of failure.

There are several other areas of additional testing that are usually considered good value on top of the AQuA testing for some apps.  Clearly it depends on a number of factors that determine the risk level.  Such factors include: the nature of the app, the market that it is intended for and the expected number of users.

Here are the main areas of additional testing over-and-above the AQuA Testing Criteria that may be worthwhile…

## Device Compatibility Testing

**Having completed a full set of tests on a single device, the next step is often to ensure that the app works across the range of targeted devices.**

Here is where we talk about the dreaded fragmentation.

AQuA's recommended practice is to carry out a smoke test on any extra devices against the smoke test sub-set of the AQuA Testing Criteria. For clarification, Wikipedia defines a Smoke Test as "a first test made after assembly or repairs to a system, to provide some assurance that the system under test will not catastrophically fail." It's a very basic set of tests that is suitable to confirm that a tested app runs (at least at a basic level) on a subsequent device. The Smoke Test can be used to confirm basic handset compatibility, but doesn't guaranteed full functionality.

Normally it is impractical and uneconomical to have full AQuA testing or full functional testing on multiple devices. However this is clearly dependent of the nature of the application. If the app is cutting-edge and stretches the device hardware then it may require more specific testing.

Typical signs of issues across devices are in the interaction with the device, and the complexity of the User Interface (UI). Special screens and special device capabilities are the most common areas of incompatibility.

## Performance testing and optimization

**Increasingly there is concern on two aspects of applications - one is data usage in a mobile environment, and the other is battery use.**

If an app is written assuming a WiFi style connection, and is careless about issues such as data duplication, caching and 'keep-alive' signals, it can use an undesirable amount of mobile data, network resources and battery life.

AT&T freely offers to developers a set of suggested best practice guidelines for development and a tool called the Application Resource Optimizer (ARO). AT&T ARO evaluates the way an app handles data usage, TCP connections, battery resources, and peripheral applications. Using the analytical data in ARO, the developer can identify and avoid issues like excess duplicate content, poorly timed data requests that can keep the device radio in a high energy state, and the inefficient use of peripheral applications like WiFi, GPS, and Bluetooth. The visibility that ARO provides into how application traffic triggers the radio on wireless networks, can improve user experience through faster response times, longer battery life, and efficient data plan usage.

GSMA also offer a series of guidelines that can help in this area.

## Regression Testing: Updates and fixes

**Regression testing is required for each bug fix or upgrade that is done to an app. It is very easy to introduce unintended side-effects (features!) with bug fixes, and the regression test will re-prove the functionality.**

If you originally used a functional test tool for running system tests on the app, it's best to re-test the app using the **same** system tests to prove that only the desired changes have occurred in the app. Usually, if only small changes or fixes have been made, it isn't necessary to re-run the complete set of AQuA Testing Criteria. A full re-run of the Testing Criteria is advisable when the changes impact a significant proportion of the app.

## Are there tools to support or automate any of this testing?

There are many tools that can support development and testing. The key is to understand which tools can help at which stages.

Sadly there is yet no single tool that will automate all the testing and answer all the needs, so developers will need to pick and choose and use a range of tools, each for specific parts of the task.

Let's start with the **Integrated Development Environment (IDE).** This will usually include a means to 'run' the app either on an emulator or sometimes by a connection to a real device. This is great for development as it allows automation of the functional testing required during development.

There are a number of tools that go further in functional test automation, usually allowing a **test script to be recorded** (either by recording the keystrokes or by some form of explicit scripting). This means a single script can be replayed many times and the output compared to a previous result. That previous result can be programmatic (a system state or API response) or graphic in terms of screen shots or videos. Such a tool is good for regression testing functionality.

There are also tools that allow **access to real devices at remote locations**. These may also accept scripts and can be used to test coverage across a range of devices for compatibility testing.

However for many tests the only reliable method is to have the device in your hand and to get the full user experience - for example, the use of vibration in a game, the use of connected accessories, or the experience of scrolling the screen.

## Are tablets any different?

Many app developers offer different versions of their app for tablets compared to the smaller screen devices. Whilst it is feasible and technically straightforward to define HTML5 or a range of screen layouts that cover all device screen sizes, it is dependent on the functionality of the app.

The use of a tablet may well be more like the use of a PC web screen rather than a small mobile device and the user experience is the key aspect that should determine whether one app User Interaction design is required for both the tablet and the smartphone or two.

Tablet-specific apps may well be more complex and as the complexity of the function of an app increases, the effort involved in functional testing must also be increased.

The rest of the testing and QA issues are largely the same, with the main difference being the network connectivity as many tablets may be WiFi only.

## Why use a test house for independent testing?

A number of companies exist to provide independent software testing services on a commercial basis: Test Houses. The test houses usually have a team of professional testers and run a variety of testing services including testing mobile apps in a variety of ways. Many developers see test houses as costly, and don't see the value of such an independent testing service.

The independence of the testing team is an important feature. They could be a dedicated test team in your own organization, or in a commercial test house. However the test team being distinct and separate from the app development team will usually provide substantial benefit in any QA process. They can provide an unbiased and fresh objective view on the app, and find problems that the developers might miss, but that a user would find. Test houses build up experience of testing apps across the market, and can use that to help you with your app.

One aspect of independent testing is the use of professional testers as opposed to developers. The mindset of a typical tester is quite different to that of a typical developer, so the testers tend to find ways to break the app that the developer would never think of.

If an app hasn't been tested by a separate in-house 'test team' AQuA strongly advises the investment of approximately $250 to have an independent commercial test house test the app against the platform-specific version of the AQuA Testing Crtieria on one device.

AQuA has approved a number of independent commercial test houses through which developers can test their apps against the AQuA Testing Criteria to gain the 'Test House Verified' quality level within the Quality App Directory.

In addition there are a number of other test houses very proficient in testing against the Criteria. Each test house may also offer further testing services that might extend the testing beyond the AQuA specifications. For example, AQuA does not specify any accessibility criteria, and some test houses offer that.

Quite clearly, different test houses specialise in different areas of mobile applications and in addition to providing the AQuA testing, they may well be able to provide advice and guidance in other areas such as localisation/internationalisation as well as specialised QA practices and skills.

And of course commercial test houses will often have a wide range of devices available for compatibility testing and will be able to offer advice on the top devices that should be covered by your app depending on the intended territory and market segment.

## How about testing for malware and viruses?

Testing an app will not guarantee that it is free from malicious software or viruses. For example the malware may be triggered by some future date, condition, network or indeed any trigger that the author may choose.

In reality, it is impossible to verify that an App is free of any form of malware. All virus scanners work on detecting already known patterns of malicious code, so anything truly new will never be caught. Fortunately most viruses and malware are variants on existing patterns and so are caught.

Malware inside an app can come from a number of sources:

- Deliberately inserted by the code writer, either with malicious intent or for innocent purposes
- Accidentally included through use of third party libraries or code fragments
- Added to an original piece of code by the action of a third party
- An accidental side effect from well-intentioned code (a Bug or a Feature!)
- Virus scanners look for patterns of known malware, and as such can detect existing known attacks, but can never find new techniques before they are identified in the lab.

Some code scanners can look for suspicious patterns of behaviour in some technologies, but this requires human interpretation.

AT&T freely offers to developers, through the Application Resource Optimizer (ARO), an integrated security scan capability provided by Lookout Security, which offers the ability to scan apps for potential security threats. The "Lookout Security" option on the Tools menu in ARO allows a developer to submit an APK file to Lookout Security where it will be scanned to inspect the components for mobile threats. The results of the scan are displayed in ARO, and the developer can save and view a history of their app submissions.

## Background on testing: terms and ideas

There are lots of terms used to define different aspects of testing. Traditionally there is little agreement on what to call what, and on what testing should be done, and the different definitions of types and objectives of testing in software development. The most common definitions are from the SWEBOK guide (SWEBOK is the Software Engineering Book of Knowledge) and a set of standards from the IEEE and the ACM).

SWEBOK defines certain testing stage during development:

- Unit Testing (or component testing) - Tests the function of each component
- Integration testing - Tests the interfaces between components and the interoperability of external components (such as a social network, a server or cloud based database or perhaps an ecommerce engine)
- System Testing - Tests the functionality on the platform

And after the coding and development is completed:

- Quality Assurance testing / Acceptance Testing -External functional testing by the user or the user's representatives or the developer or developer's QA/test team.

During development the Unit Testing, Integration Testing and System Testing tends to focus on **Functional testing** (e.g. the retrieve stock price function in a finance app) as opposed to **Non-Functional testing** which is focused on other aspects, often environmental, such as application stability.

**Functional testing** refers to checking a specific action or function of the code against the application requirements documentation, use cases or user stories. Functional tests tend to answer the question "can the user do this" or "does this particular feature work."

**Non-functional testing** refers to aspects of the software that may not be related to a specific function or user action, such as stability, security, performance or other behavior. Non-functional requirements tend to be those that reflect the overall quality of the app, particularly in the perspective of its users.

After development is complete is the time when there may be tests of the complete app including the non-functional aspects. This may take the form of a QA test or an Acceptance test. Acceptance testing can be carried out by the developer or by the end user. Typically if an app has been commissioned the purchasing commissioner will do an acceptance test before paying for the development.

**Acceptance testing** or **QA testing** will usually consist of running through a test script and test plan that will run through a set of tests designed to fulfill certain testing objectives. The principle difference between the two is that QA testing is done by the developer and acceptance testing is done by the customer. Clearly the wise developer would also carry out the same acceptance test prior to delivery to the customer.

Whilst a strict definition of Software Quality Assurance encompasses the entire process of producing an app – from requirements through design, development, testing and release – we will focus on one small aspect of that cycle, the testing of the complete app after development and prior to release to the customers.

**QA testing** should be a well defined series of tests that are run and passed after the app has left the hands of the person who coded it, but before it is delivered to a customer or shop (distribution channel).

A good QA testing process at that stage of the lifecycle will ensure both that the coder has hit the functional requirements but also met the non-functional needs. It will help ensure that when the app is submitted to a distribution channel that has technical acceptance tests, it should pass first time.

And then there are tests carried out while the app is in production; **Alpha testing**, and **Beta testing.** This sort of testing should be focused on trialing the functionality, in essence proving the requirements, rather than testing the implementation.

And finally there is the ultimate proof which is "in-life" user feedback, bug reports and updates providing continuous feedback from the users.

## Other types of testing that can extend confidence in the app

SWEBOK defines other testing objectives and types of testing including:

**Installation Testing** - to ensure that the app installs onto the target platform(s)

**Compatibility testing (Device Compatibility testing)** - to ensure that the app works with the target platform(s) and its' typical environment

**Smoke testing (and sanity checking)** - a quick functional test to make sure that the app runs and basically works in a variety of situations

**Regression testing** - ensuring that an update or fix hasn't disrupted previous results

**Performance testing** - tests to check that the app speed, data use and power consumption is acceptable and to guide improvements in performance.

**Usability Testing** - tests focused on the user interface and user interaction with the app.

**Security testing** - tests concerning the handling of personal or secure information

**Internationalization and localization testing** - tests for language and styles across the territories targeted by the app. (hint: Never trust automatic translation…)

**Accessibility testing** - tests for access by disabled persons (see WAI or W3C for details).

## About the App Quality Alliance (AQuA)

AQuA (www.appqualityalliance.org) is a not-for-profit industry initiative, formed by significant players in the mobile apps market who realized that they could benefit from pooling their knowledge in where apps go wrong.  The core members of AQuA have developed platform-specific Testing Criteria and Best Practice documents, based on their experience, to find and prevent the most common errors often present in mobile apps.

**The AQuA Testing Criteria and Best Practice Guidelines**
We developed (and continue to develop) sets of AQuA Testing Criteria which are designed to look at the effectiveness of the implementation of an app, helping to ensure that the developer's implementation is free from the common undesired bugs, and effectively delivers what the developer intended for the user to experience.

They include tests that compliment the functional testing carried out by the developer during development, covering a range of non-functional, usability, functional sanity test, installation, performance and compatibility testing. They're not aimed at quantifying the value of the idea behind the app, or the relative merits of one app over another in terms of the designed user experience or function.   Such choices are best judged by market acceptance and the success of the app relative to its competitors.

The range of tests has been derived from the industry experience of the members, developers and test house to find the majority of the common errors with a ‘right sized’ amount of testing.

Within the criteria, there are sets of tests that can be used as smoke tests for compatibility testing across a range of devices.

There’s also a set of platform-agnostic **best practices** that can be used in the design process of an app to compliment the functional design.

**The AQuA Online Testing Tool**
This is an online tool – available since February 2013 – enabling developers to run through the tests within the AQuA Testing Criteria for Android Apps online, assessing the results along the way.

**The AQuA App Quality Directory**
In October 2012 we launched a Quality Application Directory that allows developers to upload details of apps that have met the AQuA Testing Criteria thereby demonstrating a commitment to quality to any parties interested in commissioning or using their apps. The Directory is being continually enhanced as a result from industry input. Starting with the Android platform, the intention is to expand it to other platforms.

www.appqualityalliance.org
www.qualityappdirectory.org
www.javaverified.com

@AppQuality