Test Strategies for Mobile Device Applications
This white paper was originally written in 2010 by Macadamian QA team members Tanya Dumaresq and Matt Villeneuve. Since that time mobile devices have evolved immensely and with that so must the strategies that we employ to test mobile applications. The content that follows has been updated with considerations relevant to present day (2017).
Why We Wrote This Paper

Application development for mobile devices has increased immensely in the last decade and is expected to continue growing at a profound pace. According to eWeek.com, 4.8 billion people were using mobile devices in 2015 and that’s predicted to grow to 5.5 billion people by 2020 (70% of the expected population!)¹.

All these users of mobile devices—like smartphones, tablets, and even e-readers—create a never-ending demand for more and more unique and useful mobile applications.

The arena for these applications is vast, from the healthcare world—where patients can now practice self-care directly on their smartphones—to social networking to the Internet of Things—which had an installed base of 15.4 billion devices worldwide in 2015 and a predicted growth to 30.7 billion devices in 2020 and 75.4 billion in 2025.²

Users want mobile applications to be simple and fast. Just one nagging bug or usability issue can spoil the entire experience. With so much competition in this space, if users don’t have an excellent experience with your application, they will switch to a rival product faster than you can say “There’s an app for that.” Vendors simply can’t afford to go to market with an application that might have a critical bug or usability issue.

The strategies presented here are intended to highlight some of the areas where the testing of mobile device applications differs from testing desktop applications. It is important to plan a test strategy that is mobile-specific, or you may overlook crucial areas of testing like how network connectivity (or lack thereof) affects your application, how screen resolution and orientation changes can spoil a user’s whole experience, or whether your application jives with what users of a particular device have come to expect.

¹ http://www.eweek.com/networking/more-users-more-devices-mean-mobile-traffic-will-surge-cisco-says

What You Can Expect

For simplicity, we divide the paper into the following main areas:

- Mobile application types
- Cross Platform testing
- Functional testing
- User Interface testing
- Localization testing
- Accessibility testing
- External Factors testing
- Performance testing
- Security testing

Be aware that testing strategies will depend on the software requirements of the application. Not all testing strategies described in this document apply to all applications. As you are creating your test plan, please take the suggestions discussed in this paper into consideration.

Mobile Application Types

It’s important to understand the type of application that you are testing and create your test strategy accordingly. There are 3 types to consider: Native, Web, and Hybrid.

<table>
<thead>
<tr>
<th>Native Apps</th>
<th>Web Apps</th>
<th>Hybrid Apps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single platform affinity</td>
<td>Cross-platform affinity</td>
<td>Cross-platform affinity</td>
</tr>
<tr>
<td>Written with platform SDK</td>
<td>Written with web technologies (HTML, CSS, JavaScript, or Server-side (ASP.NET, etc.))</td>
<td>Written with web technologies (HTML5, CSS3, and JavaScript)</td>
</tr>
<tr>
<td>Must be written for each platform</td>
<td>Runs on web server, viewable on multiple devices</td>
<td>Runs locally on the device, supports offline</td>
</tr>
<tr>
<td>Access to all native APIs</td>
<td>Centralized updates</td>
<td>Access to native APIs</td>
</tr>
<tr>
<td>Faster graphics performance</td>
<td></td>
<td>App store distribution</td>
</tr>
<tr>
<td>App store distribution</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Native apps

Native applications on multiple platforms must be tested and certified for each platform. It is a misperception that successfully testing the app’s functionality on one device provides assurance across all devices of the same operating system (OS). Native applications are tied to the hip with the hardware and operating systems for which they are written. It is essential to test on the physical devices supported by your application, and for both backward and forward compatibility with each older generation of the device your application is expected to support as well as on devices made by different manufacturers. You’ll also want to account for updates to the operating systems, especially popular “major” releases that users will apply quickly upon availability.

Web apps

Web applications can be tested in the same way that desktop web applications are tested. In this case, compatibility testing is important to ensure that the application provides the same functionality and user experience across multiple browsers and browser versions, as well as across devices with various screen sizes and resolutions.

Hybrid apps

Hybrid applications leverage server-side components which can be functionally certified for one platform and then should work for all the platforms.

The functional testing of a hybrid app where Web parts are involved can be carried out on browsers such as Chrome or Firefox. Both Chrome and Firefox come with additional plugins that can be used to create custom user-agent headers that mimic the mobile user-agent. Browsers can also be used to test Web pages for different screen sizes and form factors. This way, functional testing can be achieved easily on the browser and to a large extent can be automated by using tools such as Selenium.

The UI/UX manual testing effort remains the same for both native and hybrid apps.
Cross Platform Testing

Device selection

A major challenge with mobile testing is achieving sufficient test coverage of a vast span of device models, OS versions, and screen sizes. In order to develop an effective mobile testing strategy it is important to determine the proper subset of devices for testing. This involves identifying the most efficient mix of devices to test based on factors such as estimated time to market and target audience.

Two sets of devices should be identified:

1. **Primary devices**
   - Physical devices to be used for manual testing.
   - Include devices from each supported platform (for example, if your application is compatible with Android and iOS then select 2 android devices and 2 iOS devices.)
   - Models and OS versions should be determined by what are currently and/or are estimated to be the most widely used at the time of release among the target audience. There is useful data out there to help determine this for various operating systems including iOS and Android.
   - Devices should have varied screen sizes to help facilitate more effective UI verification.
   - Thorough Functional, Exploratory and Regression testing should be performed on these devices throughout each project sprint.

2. **Secondary devices**
   - Real devices accessible via a cloud based tool and/or emulators.
   - Include a diverse set of devices also predicted to be used by the target audience but at less frequency.
   - Sets of UI, Performance, and Regression tests may be executed on these devices towards the end of each project sprint.
Minimum supported OS versions

Data obtained directly from operating system developer dashboards may be used to determine the minimum supported OS versions on a project. It is ideal to aim at supporting 90% of active devices.3

An iOS example: According to data measured by the Apple App Store and published on the Apple Developer Dashboard on September 12, 2016, usage is: 88% iOS 9, 9% iOS 8, and 3% earlier versions. iOS 10 was released after this data was accumulated. Based on this our suggestion is to make iOS 9 the minimum supported version and to include a minimal number of iOS 8 devices as secondary devices.

An Android example: According to data collected by the Android App Store and published on the Android Developer Dashboard during a 7-day period ending on Sept 5, 2016, 90% of active devices are OS 4.2 and newer so our suggestion is to make OS 4.2 the minimum supported version.

Testing with emulators

Emulators can be very useful to cover a breadth of devices (with different screen resolutions for example) to which you may not have physical access.

It is important, however, to keep the following caveats in mind:

1. Not all activities can be realistically emulated, like switching network connections, or taking a picture or video.
2. Some activities just don’t work at all on emulators, like streaming video.
3. Your application could exhibit different performance when run on an actual device versus in an emulator.
4. If the emulator and actual device have different dpi resolutions, your screens may not display as you expect.

3 Identified by Android as a best practice
Because of these and other more subtle differences, be careful not to assume that just because your application works perfectly on an emulator, it will have no issues on a real device. If possible you can initially test your application simultaneously on a real device and its analogous emulator, take note of any discrepancies seen between the two and base your future real device testing plan on the differences observed.

In general, it is a good idea to use some combination of real device and emulator testing, as recommended in the table below:

<table>
<thead>
<tr>
<th>Type of Testing</th>
<th>Manual Testing</th>
<th>Automated Testing</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Using Devices</td>
<td>Using Emulators</td>
</tr>
<tr>
<td>Unit Testing</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Integration Testing</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>System Testing</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Regression Testing</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Compatibility Testing</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>GUI Testing</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Performance Testing</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Security Testing</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Synchronization Testing</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

As an alternative to both emulators and buying a lot of physical devices, you could also consider using a cloud service like Xamarin Test Cloud or DeviceAnywhere.com that gives you online access to numerous real devices on various networks.
Functional Testing

Functional testing can, for the most part, be carried out in the same way you would perform it on any other kind of application. For this reason we won’t go into specifics other than pointing out areas that are of particular importance for mobile apps. Keep in mind that functional testing should include testing all application features and not be focused solely on these areas.

- **Installation** — Install and delete the app multiple times.
- **Upgrades** — Update from an old version to the latest one.
- **Sign-up & Login** — Enter various input combinations, valid and invalid.
- **Provisioning** — Create, edit, and remove user accounts; configure security settings etc.
- **Device-specific functions** — Test areas of integration with the device camera, sensors, and communication interfaces, etc.
- **Error messages** — Invoke error messages to determine if they are clear and concise.

User Interface Testing

An important area to explore in your test strategy is the user interface. With the high use of mobile applications, there are accepted guidelines for look and feel and overall behaviour. It is your job as the tester to confirm that your application follows these principles.

Note that this paper does not cover Usability testing for mobile devices, which is critical to creating a compelling, usable mobile application.

For more on usability testing and other user-centered design principles see our collection of design & usability resources.
Comparing your application with pre-installed applications

If you are not already a regular user of the device you are testing with, the first thing you should do is familiarize yourself with the device and some of its common pre-installed applications such as Phone, Email, Camera, Contacts, Calendar, etc. These are applications that device users see everyday so it is important that your application has a similar look and feel.

There are also UI guidelines available for review from various platforms such as iOS and Android.

As you are exploring, take note of UI consistency:

1. Overall color scheme/theme of the apps.
2. Style, alignment, and color of icons.
3. Font family and font size.
4. Progress indicators when pages are loading.
5. Menus. How they are invoked and typical items they contain.
6. Spell check and naming conventions.
7. Overall responsiveness of applications on the device.

Screen orientation / resolution

If your application is supported on various devices that have different screen resolutions, make sure you test early with the device that has the smallest screen. This will flush out any layout problems where the design is simply too large to fit on the necessary screen size. Of course, don’t forget to do regression testing of any subsequent layout changes to make sure your application still looks good, and be sure to test on larger screen sizes as well.

If the application under test supports screen orientation changes be sure to include a lot of tests where you rotate the device from portrait to landscape display, and vice versa, on all of the pages within the application. This reorientation
completely changes the screen dimensions so layouts may no longer display as intended. This is also a way to test for unexpected exceptions around screen reorientation events that could cause the application to crash, especially if you do this reorientation back and forth quickly before the previous orientation change gets a chance to fully complete.

It is also important to test input reactions when the screen orientation is changed. Try using the soft keyboard while changing the orientation repeatedly.

**Touch and gestures**

Touchscreens provide numerous things to consider, such as the following:

1. **Multi-touch vs. single-touch screens** — If your device and application support multi-touch features, like the pinch-to-zoom effect on iPhone, be sure to include lots of test cases involving touching the screen in more than one place simultaneously, especially while typing on the soft keyboard.

2. **Long touch vs. short touch** — While there is usually no concept of a double-click on touchscreen devices (although there could be if specifically implemented in your application), there is often a distinction between long touches and short touches. For example, pressing and holding an item may bring up a context menu, while short-clicking the same item may automatically perform an action.

3. **Button size and position** — Ensure that buttons and icons are large enough and far enough from the edges of the screen to be easily clicked by a large fingertip. For example, iOS recommends a minimum tappable area of **44 x 44 pt** for all controls, while Android recommends that touch targets be at least **48 x 48 dp**.

4. **Workflow** — Applications should ideally make use of multiple choice selections like radio buttons and checkboxes to minimize the amount of typing the user needs to do, which can be excessively time-consuming.

5. **Other input methods’ interactions** — In most cases your device will also be compatible with other input methods such as a Bluetooth or USB hard keyboard. It is important to ensure that virtually simultaneous inputs from all the methods do not interfere with each other.
Keyboards

Pay special attention to how the user must interact with the soft keyboard. There are various special cases and corner cases that should be considered.

1. Does the keyboard automatically appear if the user’s main action is to enter some text?
2. Does the first layer of the keyboard include shortcut “@” and “.com” keys if the highlighted field is for entering email addresses?
3. Can the keyboard be dismissed and redisplayed easily?
4. Do keyboard characters entered in password fields only show up as ****?

Hard keys

Be sure to include testing around the use of the device’s available hard keys such as Power, Home, Menu, Volume, and Back. In order for the user to have a great experience, these should interact with your application similarly to how they interact with the device’s pre-installed applications.

Keyboard Shortcuts

Find out if there are any expected shortcuts common for your device and test their use within your application.

For example, you can find a great list of Android keyboard shortcuts on the web.

Localization Testing

This type of testing is typically a manual exercise focused on verifying that translated strings are rendered properly in the user interface. This type of testing can also be carried out in the same way you would perform it on any other kind of application. For this reason we won’t go too far into specifics other than pointing out some areas of consideration.
Considerations

Localization testing includes testing the localized product in accordance with national language standards, searching for untranslated text in the user interface, verifying consistency of formats (date formats, number formats, etc.), verifying the correct use of currencies, verifying accordance with capitalization rules, etc.

Major points to be considered in this type of testing are:

1. Language
2. Region
3. Dates and Numbers

An internationalized application design is one that can accommodate localized content such as the following:

- characters in non-western letters
- text expansion as in European languages
- text contraction as in Asian languages

The following areas should be included in testing:

- Content: Static and dynamic content (for example: lists, search results, labels, error messages, etc.)
- Dates: Proper translation, abbreviations
- Special characters: Different languages have different sets of characters
- Postal codes: In some countries postal codes contain letters
- Phone numbers: Different formats for different markets, North American standards
- Text Direction: Most languages are written left to right, but others are right to left (for example: Arabic)
- Currency conversions: Especially important for retailers
- Tax calculation: VAT, sales tax, and others vary from locale to locale
Accessibility Testing

Given the vast and diverse demographic of mobile device users it is important to ensure that your application is widely accessible. Guidelines have been compiled by the World Wide Web Consortium (W3C) to help with this. They are based on the principles of the Web Content Accessibility Guidelines (WCAG) 2.0 and tailored to be mobile-specific.

The following high-level areas should be taken into consideration when creating your test strategy.

**Principle: Perceivable**
- Account for small screen size: minimize amount of information displayed, adapt length of link text to viewport width, position form fields below labels
- Zoom/magnification: support up to 200%
- Contrast: sufficient contrast between text colour and background colour

**Principle: Operable**
- Keyboard control: no keyboard trap, comprehensible focus order, focus visibility
- Touch target size and spacing: large enough and easily selectable
- Touchscreen gestures: easy to carry out, elements activated via mouseup or touchend event
- Device manipulation gestures: touch and keyboard operable alternative controls available
- Buttons placed where they are easy to access

**Principle: Understandable**
- Changing screen orientation
- Consistent layout
- Positioning important page elements before page scroll
- Grouping operable elements that perform same action (example: image and text)
- Provide clear indication that elements are actionable
- Provide instructions for custom touch screen and device manipulation gestures
Principle: Robust

- Set the virtual keyboard to the type of data entry required
- Provide easy methods for data entry: select menus, radio buttons, checkboxes, automatically entering known information
- Support the characteristic properties of the platform: screen readers, switch access, display accommodations, etc.

External Factors Testing

There are numerous factors that are external to the application itself but inherent to the mobile device that the application will run on. It is important to also test how these factors may influence your application.

Network connections

Since your application is going to be used on devices in various locations with various network connection speeds, it is important to consider who your end users are going to be and to plan test coverage accordingly.

Main scenarios to test are the following:

1. Only Wi-Fi connection
2. Only 2G/3G/4G connection
3. Only LTE connection
4. No connection
5. With no SIM card in the device
6. Intermittent connection scenarios that a user might encounter in the real world:
   - Walk out of Wi-Fi range so the connection automatically switches to 3G/LTE
   - Ride in an elevator or on a train where the network connection may go up and down
7. Bluetooth connection or disruption scenarios depending on the range of Bluetooth (~100m)
You may need to do some in-depth planning for your network testing if having high/low bandwidth or packet delays are critical to the behaviour of your mobile application (for example: applications with audio/video calling functionality.)

This could include replicating network conditions using network management tools such as Dummynet in order to:

- Configure network delays
- Configure network packet loss
- Configure high or low network bandwidth
- Etc.

**SD card interactions**

If your application can potentially store or retrieve items on the device’s SD card, then it is important to test the application’s behaviour when there is an SD card present or not. At a minimum, the application should provide user-friendly error messages when a function cannot be performed due to a missing SD card.

Also consider removing the SD card in mid-operation (particularly on devices where it can be easily removed without taking the back off of the device.)

**Interruptions and notifications**

Be sure to test the following scenarios to see how your application reacts before, during and after a phone call:

1. Your application is interrupted by an incoming call, originator hangs up the call
2. Your application is interrupted by an incoming call, terminator hangs up the call
3. Your application is interrupted by placing an outgoing call, originator hangs up the call
4. Your application is interrupted by placing an outgoing call, terminator hangs up the call
Also take into consideration such interruptions as:

1. Text messages
2. Voicemail notifications
3. Calendar events
4. Social media notifications (Facebook, Twitter, etc.)
5. Alarm clocks
6. Low battery notifications
7. Forced updates

Any of the above could have an impact on the functionality or overall responsiveness of your application.

**Device options**

Explore your device’s options and change settings such as the following to see how they affect your application:

1. **Sound profiles** — Does your application respect the device’s set profile?
2. **Device password / unlock pattern** — You may be able to configure this as a requirement during application installation (for example, setting device option to prompt on Application Install if it exists.) Does your application still install correctly when prompted for password/unlock pattern?
3. **Font** — How does choosing a different font family, size, or style affect the appearance and usability of your application?
4. **Screen timeout / Auto on, off** — Is your application subject to screen dimming or automatically turning off even when it is actually busy? For example, you wouldn’t want your screen to dim or turn off while watching a slideshow of your photos.
5. **Screen orientation** — You may be able to enable/disable automatic orientation switches when the device is rotated. Does your application respect this setting? (However, also keep in mind that the application code can lock a screen to a particular orientation so if you find an issue here it might actually be design intent.)
6. **Connections** — Using one of the connections on a device, such as Bluetooth, could have adverse effects on your application. How does enabling/disabling Bluetooth or other connection types affect your application’s behaviour?
7. **Time / Time-zone change** — How does changing the phone’s time or time-zone from the settings affect the behaviour of your application?
8. **Theme** — Does changing the device theme affect the usability of the app?

9. **Multi-tasking** — Observe how switching from your app to other apps in the middle of a sync or data migration or any other kind of operation affects your application.

10. **Power saving mode** — Does your application handle this mode gracefully?

---

**Performance Testing**

Although mobile devices’ memory and power have improved significantly over the past years, mobile applications still have more memory and CPU constraints than desktop applications.

**Acceptable load time**

Load time acceptance criteria is tricky to determine. Acceptable times may vary greatly depending on such things as target audience, purpose, market competition, etc. The level of tolerance for loading times is also subjective from user-to-user.

Luckily there are benchmarks available for **iOS** and **Android** applications. According to data from February-March 2017, 57.29% of iOS applications load in <=2 seconds and 66.74% in <=3 seconds; and 50.51% of Android applications load in <=2 seconds and 55.83% in <=3 seconds.

**Load testing techniques**

Load times of Web applications can be analysed using browser tools such as Chrome Developer Tools.

Determining load times of Native or Hybrid applications is not as straightforward if you do not have Performance Testing tools readily available. In these cases an effective method is to manually video record traversal through the application, load the video into a video editing application, record the loading start and stop times of each screen, and then convert the
difference into seconds (or microseconds) taking into account the video frame rate. Repeat this process several times in order to find the average load time for each screen.

**Stress testing techniques**

With users expecting applications to hold up even under high traffic conditions with many applications running, stress testing in particular is a must to find exceptions, hangs and deadlocks that may go unnoticed during functional and user interface testing.

The following are some stress testing techniques to try:

1. Load your application with as much data as possible to try to reach its breaking point.
2. Perform the same operations over and over again, particularly those that load large amounts of data repeatedly.
3. Perform the repeated operations at varying speeds – very quickly or very slowly.
4. Leave your application running for a long period of time, both interacting with the device and just letting it sit idle, or performing some automatic task that takes a long time (like displaying a 200-photo slideshow.)
5. Randomly send screen taps and keystrokes to your application.
6. Have multiple applications running on your device so you can switch between your application and other device applications often, and at different states within your application.
7. Use all means to launch your application.
8. Open and close the application repeatedly.
9. Have a large number of users access the application at the same time on separate devices.

**Tools**

There are some tools you can make use of for stress testing on mobile devices. For example, **Monkey** can send random clicks, touches, or gestures to your Android application.

Android also has **DDMS** built into its SDK, which among other things can be used to check memory usage.
Non-reproducible issues

Another general tip related to stress testing – do not be afraid to report bugs that you haven’t yet found a way to reproduce 100% of the time. If you can at least provide crash logs and a best guess as to how to trigger the issue, that is usually enough to give the developer some ideas on narrowing down the root cause.

Security Testing

General considerations

There are a few things to consider around security at the device level when comparing mobile applications to desktop applications:

1. Most mobile devices assume one user; however, some devices may support multiple user accounts therefore the following should be considered: user switching, multiple profiles, and permissions based on user level.

2. If your application deals with the device’s Contacts or Email functions, testing should include the case of having multiple email addresses configured per device.

3. It is up to the user whether or not they configure a password/unlock pattern for their device at all.

4. If the device does not support hardware encryption then all necessary encryption needs to occur at the application level.

5. Outside communication of any sensitive data should be done over SSL/TLS because it can’t be assumed that the user will configure their Wi-Fi to be secure.

More detailed information on how to perform security testing on your mobile applications will be provided in a future paper. In the meantime consider incorporating the areas covered in the following sections into your test strategy.

Web applications

While running most web security testing tools on a mobile device can be tricky, you can run your mobile website on a PC itself, by following the instructions here.
Then you will be able to test the application with common web security testing tools, such as Firefox Web Developer add-on, Burp Suite, Zed Attack Proxy (ZAP), or WebScarab. Information on additional testing tools can be found here.

A web proxy can also be used to intercept all mobile device traffic to monitor data and test for security issues.

**Sensitive information**

The threat of passwords or usernames being left unencrypted on the device is a real one that can be taken advantage of by other users.

To test for this, footprint and fingerprint analysis can be used to make hash files of the device file structure, with a tool like md5deep. Then use a difference comparison application such as ExamDiff to compare the before and after hash files to see which files have changed. By doing so, you can then search the files your application changed for unencrypted information that could be used to compromise the device.

**Application and device permissions**

During installation of your application on a mobile device you are likely to encounter options for application execution permissions. Research the permissions configuration on your device and design your testing to ensure that your application can get access to the device areas it needs in order to function properly, and it provides useful error messages directing the user to set the permissions properly if it cannot.

The device itself could also have different security configurations, such as security off, various levels of prompting, required third-party signing, or locked. It is important to verify that your application can be successfully installed at all expected device security level settings.

Even after installation, it is possible to change the security settings of your device or application permissions. You should include some test cases to cover these scenarios as well.
Conclusion

When planning your testing effort for a mobile device application it is important to consider the following areas and how they differ from desktop applications:

1. **Mobile application types** — Testing strategies should be tailored to the type of application that is being tested: Native, Web, or Hybrid.

2. **Cross Platform testing** — Mobile devices vary between platforms, models, and operating system versions so it is essential to select a subset of devices relevant to your application.

3. **Functional testing** — App functionality should be fully tested, with special consideration given to installation, updates, sign-up and login, provisioning, device-specific functions, and error messages.

4. **User Interface testing** — Mobile devices have unique user interfaces such as smaller screens that can be re-oriented, touchscreen gestures, and soft keyboards.

5. **Localization testing** — Text length differs between languages so it is especially important to test all pages in your application given the small screen size of mobile devices.

6. **Accessibility testing** — Mobile devices have a diverse demographic of users so it is important to ensure that your application is widely accessible.

7. **External Factors testing** — Mobile device applications must also contend with interactions and interruptions from other device features like various network connection types, SD cards, phone calls, and assorted device settings.

8. **Performance testing** — Mobile device applications have less overall device memory and power available so must handle themselves very efficiently.

9. **Security testing** — Mobile device security has become more and more important as the user base grows, so it is essential to test the security of your mobile application’s sensitive data storage, and how your application behaves under various device permission schemes.
Test Strategies for Mobile Device Applications
by Sarah Savoy, Principal QA Specialist & Soseh Cyousefian is a Senior QA Specialist

About the Authors
Sarah Savoy is a Principal Quality Assurance Specialist at Macadamian. She has been working in the quality assurance field since 2005, with a primary focus on mobile application testing for the past 5 years. She is active in the software testing community holding a director role on the Canadian Software Testing Board (CSTB) and being a member of working groups on the International Software Testing Qualifications Board (ISTQB).
ssavoy@macadamian.com

Soseh Cyousefian is a Senior Quality Assurance Specialist at Macadamian with ten years of experience in mobile, desktop and web application testing. Soseh evaluates software quality with a focus on the user and the user experience because she believes in user-centered design of the software. She has been an integral part of the release management, testing and delivery processes at Macadamian. She holds a masters degree in Management Information Systems, is ISTQB certified and is currently working on developing test strategies for connected health and IoT projects.
shovsepian@macadamian.com
Thank you.

Macadamian is a full service software design and development firm. From product ideation to market ready – and everything in between, we provide a complete range of usability, design and engineering services. From big consumer brands to enterprise, telecom, and healthcare; our solutions are founded in design that thinks of the customer first while leveraging the cloud, Big Data, and Internet of Things to deliver context-aware and adaptive experiences. The result? Engaging software solutions that have a direct impact on the bottom line.

Visit macadamian.com
Call 1-877-779-6336