Testing Smart and Mobile

When will software testing be truly mobile?
Ole Lensmar

The Ins & Outs of Mobile App Testing
Jim Cowart

A Field Guide To Mobile App Testing
Rosie Land

The Internet of Things: Software Testing's New Frontier - Part 2
Michael Hackett, LogiGear Corporation
Letter from the Editor

I led the Editor’s Note in our very first mobile issue with “Everything is mobile”, but it is now way beyond what we thought. Mobile has come to mean only the smartphone, mobility is the word that describes everything a smartphone enables you to do.

Mobility is more than a device! Mobility is business infrastructure that allows accessing data and services from a host of devices. It’s using a smartphone as the remote control for the Internet of Things (IoT). It is context awareness of your device based on location or time, or just about any other data point. Mobility is Google Glass, Apple Watch, remote control for your nanny cam or home security system. Mobility opens up testing projects to enterprise data, apps, analytics, and cloud services.

The most exciting part of mobility for testing is the amazing array of products and services we have to test. Mobile testing now means testing controllers for the IoT. It means testing headsets with tiny screens and gestures for input. It’s testing data and security, not just for a device but of an entire connected, mobile system.

The awareness, strategy, knowledge and skills needed for this greatly expanded understanding of mobile is huge. Testing context aware access and information based on sensor information from geolocation and time is a whole new level above a mobile-enabled website.

Mobile platforms have exploded—in a precarious way for test teams. Compatibility, integration and security testing are different animals today. The use of smartphones as the remote controls for so many products and services has elevated them to personal command centers. Sensors and the explosion of the IoT has created large areas for learning and tool use with emulators/simulators, and communication protocols. The testing ramifications are interesting and the business demand for rapid development adds pressure, more intense need for great communication and risk analysis.

In this issue we cover a broad swath of today’s mobile testing environment, from the mobile portion of the IoT, to what types of issues to be aware of, how and what to test, and mobile test automation. My hope is this issue of LogiGear Magazine will help you with your mobile testing.

Our next issue will include getting help for your test effort; testing professional services, outsourcing, and offshoring, and we’ll also announce 2015’s editorial calendar.
In this Issue

4 IN THE NEWS

5 BLOGGER OF THE MONTH: WHEN WILL SOFTWARE TESTING BE TRULY MOBILE?
Ole Lensmar
Will testers bet among the first IT professionals to shift their toolset to tablets and smartphones?

7 COVER STORY: THE INTERNET OF THINGS: SOFTWARE TESTING’S NEW FRONTIER - PART 2
Michael Hackett
This the second part of a two part article that analyzes the impact of product development for the internet of things (IoT) on software testing.

15 BOOK REVIEW: PENETRATION TESTING - A HANDS-ON INTRODUCTION TO HACKING
Xavier Mertens

17 FEATURE: A FIELD GUIDE TO MOBILE APP TESTING
Rosie Land
Great mobile testing requires creativity to find problems that matter.

23 FEATURE: THE INS & OUTS OF MOBILE APP TESTING
Jim Cowart
Removing the barriers to more and better mobile testing.

26 FEATURE: THE AUTOMATION OF MOBILE TESTING – SHOULD WE OR SHOULDN’T WE . . .
Sam Mullin
Manual testing teams may not be able to test all the processes with each build.

28 TOOL PREVIEW: TESTARCHITECT MOBILE PLUS
LogiGear Staff
A fully integrated test automation platform for mobile testing.

30 MOBILE TESTING GLOSSARY
Some of the terms used when discussing mobile testing.
According to Gartner most mobile apps fail security tests

A new report from researcher Gartner claims that 75 percent of mobile apps won’t pass the most foundational of security tests. That means enterprises accommodating a mobile workforce with BYOD (bring your own device) policies in which employees can access sensitive information or carry out normal business activities may inadvertently compromise networks and expose confidential data. Dionisio Zumerle, a Gartner principal analyst, stated that testing the mobile applications code and graphical user interface doesn’t go far enough. Enterprises must secure the servers that communicate with mobile clients to access a business’s applications and databases as well.

Source: thevarguy.com

Negative reaction to proposed testing standards

The International Standards Organization, in collaboration with other standards bodies, is preparing a new standard for software testing: ISO/IEC/IEEE 29119. This standard is intended to codify many of the practices and techniques of software testing. The response from the testing community has been largely negative due to the prescriptive nature of the standard, and the exclusion of the context-driven approaches. The new standard will replace existing standards previously published by a number of different contributing bodies. James Christie gave a talk at the CAST 2014 conference in which he rejected the premise on which the certification is based. He wrote a blog post titled “ISO 29119: Why it is Dangerous to the Software Testing Community” in which he expanded on the ideas from his talk.

Source: InfoQ

New Jersey requires testing to certify gaming software

The process through which regulated online poker and casino software is tested in New Jersey is technical, complex and largely removed from public view. The Division’s Technical Services Bureau (TSB) employs mathematicians, electrical engineers, programmers, and IT Security Specialists to evaluate the products’ random number generators, disclosure of the rules of the game, game play, expected return to player (RTP), general fairness and integrity. The Bureau, as part of the approval process for poker, uses volunteers from various bureaus of the division who are familiar with poker rules to evaluate the game play and fraud detection. Specialists are used to conduct the technical evaluation.

Source: http://www.onlinepokerreport.com
When will software testing be truly mobile?

Will testers be among the first IT professionals to shift their toolset to tablets and smartphones?

By Ole Lensmar

As I’m sure you already know, a monumental shift from desktop to mobile is upon us. Not only have consumer applications started leaving the desktop behind, but B2B applications are also starting their migration – like a flock of elderly pelicans, they spread their wings to follow the younger seafowl. And although it still might be hard to envision a tablet version of your favorite word processor or spreadsheet, rest assured that someone will spearhead that shift, using a mobile-inspired touch-driven UI with all the bells and whistles the mobile experience makes possible, to rescue word processing or spreadsheet from the grey and aging cobwebs spreading over your desktop.

With my developer hat on, I can’t wait for this shift to happen for everyday development. When I’m into some coding project – be it big or small – my mind is constantly occupied with solving hands-on coding problems. How should I structure my code? What feature should I add next? How am I going to squash that bug? What sorting algorithm should I use? Being able to solve those problems immediately and easily on a tablet and “on the go” would be a dream come true – the spontaneous and creative nature of development nurtured to full bloom. Of course, the actual processes for compiling, running integration tests and executing builds might well be on a server in the cloud somewhere, which is even better as it would allow me to pick up on my line of thought from wherever I am, whenever I want to.

The same should apply for testing; as many passionate testers will tell you, testing for them is just as creative a practice as coding is for me. They carry their tester curiosity with them at all times, always thinking about how to challenge their target applications and break them. To me, they seem to be a prime target for a mobile mini-revolution; not only should they be testing mobile apps, the whole quality lifecycle should be available at their fingertips on...
their mobile devices: test design and management, exploratory testing, test recording and execution, regression testing, etc. It even makes extra sense for testers considering the fact that more and more applications have a mobile component. They have to be tested “in the wild” with fragile networks, bad positioning signals and draining batteries. Empowering testers with the ability to perform their testing (be it automated or exploratory) in the same environment as the end user - on trains, in tunnels, in cities, in the country, etc. – is extremely valuable because this is usually where things go wrong in the end, and not in your test lab at the office.

So will it happen? Will testers be among the first IT professionals to shift their toolset and workflows from desktops and laptops to tablets and mobile devices?

Unfortunately, probably not. Testing and Quality Assurance as a whole seems to be a conservative domain, both from a tooling and process perspective, and testers as a group have often been slow to adopt many of the ongoing trends in development (agile, automation, DevOps, etc.) – their adoption has been more reactive than proactive. To a certain extent this “coming late” is perhaps attributable to the tester mindset -questioning and probing before embracing. Unfortunately, I also think testing as a profession and practice has generally been kept short in larger organizations and not seen as a target for investment and innovation.

Another hurdle for this shift is the anti-tool movement within testing. Many testers refrain from using tools in general, as they don’t want to be “trapped” in a tool-imposed line of thought. Many testers feel (with good reason!) that tools hamper their creativity and out-of-the-box mindset which is so essential to successful testing. They have a point; you should be in control of the process and tools, not the other way around. But since testers are also the ones driving tool creation, and given how uncharted this territory is, testers could see mobile as an opportunity to build tools the way they want or need them - unobtrusive, modern, dare I say “agile”?

Perhaps this is the opportunity to propel testing and testers to the forefront of software technology – no more backwaters of VBA macros and archaic scripting languages. Bring on the touch interface to capture and facilitate the graceful art of testing all around us – everywhere – at all times!

Ole Lensmar is chief technology officer at SmartBear Software, allowing him to live his passion for software development in a creative and thriving work environment. Ole is the co-founder of Eviware Software which was acquired by SmartBear in 2011.

(This article was originally published on http://www.network-world.com/article/2225214/opensource-subnet/when-will-software-testing-be-truly-mobile-.html)
The Internet of Things: Software Testing’s New Frontier - Part 2

This the second part of a two part article that analyzes the impact of product development for the internet of things (IoT) on software testing.

By Michael Hackett

Remote control

Remote access is nothing new to embedded systems. From HVAC systems, to medical devices, to factory automation, embedded systems have been used to monitor, change setting and turn things on or off from a remote location.

Many of us use IoT devices with remote services without thinking about it as remote control. Digital video recording set-top boxes, such as a Tivo device, use remote control for downloading and playing movies from a browser, not only from home, but anywhere there is internet access. And it’s now common to use mobile phones to operate video cams to check on the baby, grandma or even the dog. It’s also hard to imagine the things you can’t do remotely, and remote interactivity is a key selling point for IoT devices.

Remote control of devices present us with three main testing issues: Interoperability, compatibility and security. Interoperability issues cause people to have specific remotes for each device, which pretty much defeats the idea of ease and usability. But this also creates opportunities fast moving players in the market to grab market share.

Jean-Louis Gassée (Apple initial alumni team, and BeOS co-founder) recently predicted we will likely have the problem he calls the "basket of remotes", where we’ll have hundreds of applications to interface with hundreds of devices that don’t share protocols for speaking with one another. Cooperation among devices and remote controls seems too much to ask for at this point. This means that supported and not supported devices and platforms must be clearly defined then tested to agreed upon coverage under the range of conditions the remote control can operate.

There’s also the dependency issue. Your embedded device may be working fine, but a remote control, or smartphone under low battery conditions may automatically turn off a sensor, or reduce functional-
ity to conserve power. The result is that your device does not work as expected. Testing remote control devices for loss of signal conditions and a host of interrupt conditions may be a bigger testing job to scope and execute than testing your limited function device.

Wearables

The word wearable in the IoT can mean many things. Wearables are not tested as a group, rather their testing implications are defined by the device type:

- Wearable can mean Google Glass, which is a complete internet enabled device by itself.
- It can mean a multifunction Apple Watch or a limited function “smart” watch that can only receive notifications but take no action. Most smartwatches currently available need another device, such as a mobile phone for remote control or a UI to other functions.
- Wearable can also mean an embedded system by itself that operates with remote control or M2M in a different way, such as accelerometers and pedometers that measure movement and collect data for physical therapy.

On the surface, testing wearable devices in the IoT is just like testing any other similar device:

- Individual sensors are tested as individual sensors— wearable or not.
- Watches with sensor functionality and a very limited UI can be tested from a mobile device.
- Wearable as command center can be tested as such.

When discussing wearables, it is important to remember one of the key aspects of handheld mobile device testing: haptics. These are the various communication sensations from the device that you feel like a click, a vibration, or a pulse. We know from mobile device testing this limits the use of emulators/simulators— an emulator cannot vibrate in your hand! Many wearable devices have haptic sensations that must be tested using the real device for function, multiple functions with perhaps multiple sensations in sequence, usability, and timing/race conditions.

Headsets like smarthelmets for firefighters for example, often include sensors for temperature, motion, geolocation, and gesture. With a camera and a variety of communication sensors there is also a display. These systems carry out common computer tasks, under uncommon conditions all hands-free via WiFi and Bluetooth. Wearables as command centers are growing very rapidly in public safety, construction and industrial application. They present interesting and complex test conditions and scenarios to create or simulate.

Range

With mobility, range becomes an issue. We know mobile 3g and 4G has to be tested under various signal strength conditions as well as loss of signal. The same goes for wi-fi. For other technologies there are often very different cases to test. For example,
each Radio Frequency Identification (RFID) chip has a defined range from 10cm (the maximum range for Near Field Communications—NFC) to 100 meters or more with certain chips.

RFID tags, readers/ transponders come in many varieties. Some in “smart cards” used similar to NFC, to long range RFID tags like those used in toll collection systems that can be read at 300 feet/~100m (the read ranges depend on power source and antennae, among other factors).

RFID tags are the most common internet of things sensors. They are already called “the duct tape of the digital age.” RFID tags are attached to cash, boxes, clothing, possessions, and even implanted in pets and people. There is a large variability in the devices capacity and function. You can’t expect to test one RFID chip in and out of range and expect to satisfy any type of test requirement.

Note: Emulators and simulators are similar but different- for convenience relating to testing the IoT, I will use simulator only.

These software tools make it possible to test situations that would be nearly impossible to replicate consistently using actual devices. A few good examples are testing for any fault or interruption and what happens when signal is lost during data transfer. From a testing perspective, simulators can be used for internal tests of base functionality, but should definitely not be used to sign-off any critical application.

Simulators can greatly facilitate early testing and may be the only window into a device with no UI. While this makes them extremely useful it’s necessary to be aware of their major limitations:

- The emulator software itself may have bugs.
- Emulators may not (likely don’t) replicate all the features as the production devices.
- The Haptic experience.
- Race conditions, asynchronous, and nondeterministic events may be impossible to simulate.

For some devices, a simulator may be the only way to automate testing. And this can lead to the situation where it’s necessary to create the simulation software. It’s not uncommon for this to take as much time and effort as building software for the actual device, and a reason some companies are reluctant to make the investment.

Simulators are extremely valuable for mobile device testing due to the sheer number of devices in the market. You can quickly test a lot of versions by automating the tests and running simulators on multiple machines or virtual machines. Obviously, testing actual devices is ideal, but it’s not always
practical, especially when it comes to Android. Android is open source and there are a lot of variations that will need to be tested. Although a generic Android simulator can be used, its best to use a simulator supplied by the device manufacture.

The thing to keep in mind is that a simulator is an approximation of a device. The key to simulators being helpful in your test strategy is knowing when they are really useful and also knowing their limitations and drawbacks.

Devices with embedded systems have been around for quite a while. Even the new sensors being developed aren’t much different than their predecessors. It’s when all these devices socialize across the internet that things get into high gear fast. Issues of performance, security, communication protocols with various bandwidth, low power and a myriad other events that have the potential to cause headaches—will at some point.

There are many common communication protocols a device may use. Some devices will use multiple technologies like MQTT for low power messaging, TCP/IP and Wi-Fi for remote control from a mobile phone or remote mainframe, cellular (#G or 4G), RFID, Bluetooth for low power device communication and more. Any of these may be required to work in a low power, limited memory environment that will be placed behind a concrete wall. How about testing all this connectivity? It is a lot!

How can you test the reliability, scalability, power efficiency of the chosen topologies and wireless protocols and radio technology? You will need knowledge, tools, and probably help. You’ll also likely need to automate some tests on a simulator. You can expect to miss bugs since there are no tools to test directly on the device itself if it has a unique OS. Maybe your team will need a lot of help. If this is the case, remember reporting risk is a foundation skill for every test engineer.

**Performance Testing**

As millions of new devices get attached to the internet, it is obviously increasing the infrastructure burden. Load, stress, failover and performance testing are required. I find that *lip service* is paid to performance testing but commitment is lacking. Surveys of CIO’s bear this out. Many say performance testing is *seen* as important but not being done. So performance testing becomes a *nice to have*. This is a problem. Performance and security testing is not free. It takes skill, tools, commitment and time. Many companies choose to take their chances with performance and/or security of their product, and then hope for the best— but hope is not a strategy.

The new crop of internet connected embedded systems requires a more advanced level of performance testing. For a variety of reasons performance testing may now be the starting place for a device test project instead of merely *nice to have*: 

**Connectivity and Communication**

Devices with embedded systems have been around for quite a while. Even the new sensors being developed aren’t much different than their predecessors. It’s when all these devices socialize across the internet that things get into high gear fast. Issues of performance, security, communication protocols with various bandwidth, low power and a myriad other events that have the potential to cause headaches—will at some point.

There are many common communication protocols a device may use. Some devices will use multiple technologies like MQTT for low power messaging, TCP/IP and Wi-Fi for remote control from a mobile phone or remote mainframe, cellular (#G or 4G), RFID, Bluetooth for low power device communication and more. Any of these may be required to work in a low power, limited memory environment that will be placed behind a concrete wall. How about testing all this connectivity? It is a lot!

How can you test the reliability, scalability, power efficiency of the chosen topologies and wireless protocols and radio technology? You will need knowledge, tools, and probably help. You’ll also likely need to automate some tests on a simulator. You can expect to miss bugs since there are no tools to test directly on the device itself if it has a unique OS. Maybe your team will need a lot of help. If this is the case, remember reporting risk is a foundation skill for every test engineer.

**Performance Testing**

As millions of new devices get attached to the internet, it is obviously increasing the infrastructure burden. Load, stress, failover and performance testing are required. I find that *lip service* is paid to performance testing but commitment is lacking. Surveys of CIO’s bear this out. Many say performance testing is *seen* as important but not being done. So performance testing becomes a *nice to have*. This is a problem. Performance and security testing is not free. It takes skill, tools, commitment and time. Many companies choose to take their chances with performance and/or security of their product, and then hope for the best— but hope is not a strategy.

The new crop of internet connected embedded systems requires a more advanced level of performance testing. For a variety of reasons performance testing may now be the starting place for a device test project instead of merely *nice to have*:
- Device limited network bandwidth,
- Interrupts,
- Low battery,
- Timing and scheduling of real time systems (RTOS),
- Transmission of large data sets,
- Intermittent connectivity,

Performance testing is not incidental. It is a separate concerted effort with different tools, different skills, SLAs (service level agreements) and benchmarks that come from other teams (marketing, legal, sales, customers, competitors, etc.).

Failover or DR (disaster recovery) testing for both hardware and software (when a service or the system go down, what happens?) is critical. Failover testing will need many varied scenarios including soap opera (rare, boundary, large data, once-in-a-lifetime condition) testing.

Borrowing from game testing, there is an additional performance test type called soak testing. Most of the IoT devices being developed are planned for long term use without powering down or rebooting. Soak testing runs the system for extended periods without stopping, restarting, rebooting, just like the real-word scenario.

Testers are not the ones who can drive more performance testing, but you do need to stay vigilant about risk assessment and communicate the issues.

Data: collection and transmission

A big part of the IoT is the collection of data. So much is written these days about the connection of the IoT and big data. Whether to give context aware information, forecast behavior, recognize patterns or optimize performance, using large sets of data with other large sets of data—such as correlating home heater and air conditioner use to ambient temperature and other weather data from the National Weather Service—the comparison and manipulation of this big data can, clearly provided giant economic, communal and environmental benefits. Big data means tester skills for database testing, data integrity testing, database performance testing, and testing cloud APIs is essential.

Security

For test engineers, security testing has a few facets. First, there are generally requirements about security, views and access. Functional testers normally validate and test these in daily work. Intrusion and penetration testing—protection from hackers—is usually done by different teams with different knowledge, skills and tools. Both types are absolutely necessary. Like performance testing, many surveys show organizations are not doing (adequate) security testing. There are well-known and sad stories here already. I’m now concerned about who’s in my wallet since the last trip to Home Depot.

Today’s new cars are chock-full of computer chips, sensors and nanotechnology having up to 100,000
lines of software code. From financial headlines July 15, 2014: "Compared with cars and trucks of a decade ago, our cars and trucks are staggeringly complex," said Lisa McCauley, vice president and general manager for Battelle Cyber Innovations, a nonprofit research organization.

Andrew Brown, Delphi vice president and chief technologist, said it would be difficult for a hacker to break into a car’s infotainment systems remotely. But for a hacker who is familiar with the software it’s a different story. "If you can get that sort of access, almost anyone could break into the system," Brown said. "Without that, it is very difficult." Let’s hope.

The U.S. National Intelligence Council predicted: “to the extent that everyday objects become information security risks, the IoT could distribute those risks far more widely than the Internet has to date.”

Security experts this month tested 275 Apple iOS- and Android-based mobile banking apps from 50 major financial institutions, 50 large regional banks, and 50 large U.S. credit unions. Overall, they found that eight out of 10 apps were improperly configured and not built using best practices software development. Among the big-name banks whose mobile apps were tested by security firm Praetorian include Bank of America, Citigroup, Wells Fargo, Goldman Sachs, Morgan Stanley, Capital One Financial, and Suntrust Banks. Praetorian did not disclose how each bank’s apps fared in the tests.

Nathan Sportsman, founder and CEO of Praetorian, says the security weaknesses in the mobile banking apps he and his team tested are not pure software vulnerabilities, so they are relatively low-risk issues for exploitation.


Weak Security In Most Mobile Banking Apps, 12/12/2013, Kelly Jackson Higgins

Information Week: Dark Reading

**Interoperability**

Recently at the Apple Worldwide Developers Conference, Tim Cook, Apple CEO pointed to iOS as the platform of choice for the IoT and highlighted the continued fracturing of other mobile platforms [read: Android] as the reason why.

After the hype clears, when discussing the IoT, the next topic is the lack of standards. This is mainly the
result of competing and uncooperative product companies all vying to be king of the market. We are stuck with this for a while and it is not a good thing and makes testing more time-consuming in tight schedules, more complex when it need not be and have us make difficult test coverage suggestions.

There are competing standards for all aspects of devices from power to internet connectivity to security. Sensors often have a bad time talking to each other. There are a few common sensor specifications and the standards are not secrets. Sensor standards using IEEE 1451 specifications have been routinely proposed – but the industry has so far chosen to do the easiest, most convenient and cheapest thing that later causes functional interoperability problems, rework, scalability and performance problems.

Test teams will not solve these problems, but we do have to test for interoperability. This is difficult enough when it comes to a traditional embedded device working with other devices. Again, once you connect to the internet, you open the door to many more issues. Multiple connectivity protocols, cloud API interoperability, remote control from a desktop browser, tablet, smartphone, watch, car... you see where this is going.

So what to do? First- find out what interoperability programmers are planning. Next find out what your marketing or sales team is “supporting” or compatible with. Then, document your testing scope and plan for a lot of testing.

**Summary**

There is no magic answer for how to test the IoT. It is complicated with many unknowns, and there are some very unique test considerations. All of this is on the cutting edge and it’s exciting.

By adding internet connectivity, databases, Big Data and cloud services to your experience base you are building skills that will be relevant for the long-term. Learn and grab information. Build skills in a big variety of test types discussed here.

It's currently a kind of “Wild West” mentality with very few standards. Many platform providers think they are already king-of-the-hill and place little real focus on performance, security and interoperability. This will undoubtedly change over time. But for now you are testing in uncharted waters.

Test early, test often. Leverage automation as much as you can. And, of course, you will be testing under tighter and tighter product development deadlines and budgets. Test as much as you can and for my opinion, report risk and coverage limitations even more than you report what you have actually tested.

Michael Hackett co-founded LogiGear in 1994 and leads the company’s LogiGear University training operations division, setting the standard in software testing education programs for many of the world’s leading software development organizations.

Michael is co-author of the popular Testing Applications on the Web (Wiley, 2nd ed. 2003), and has helped many clients produce, test and deploy applications ranging from business productivity to educational multimedia across multiple platforms and multiple language editions.
WEARABLE TECHNOLOGY

Glossary

Smart Glasses such as Google Glass have begun shipping.
Heads up displays for gaming include Microsoft Kinect.
Market shifts in activity monitors market - Jawbone buys BodyMedia.
Pebble Smart Watch begins shipping.
Stretch and bend sensors in clothing increasingly available.
Sony, Samsung and Qualcomm launch new smart watches.
Apple files patent for smartwatch technology.
Microsoft files patent for smart glass technology.

WEARABLE TECHNOLOGY DEVELOPMENTS

Investors have spent over half a billion dollars in equity and debt on wearable tech startups since 2009.
The global wearables market was worth more than $2.5B in revenue in 2012 and is expected to cross $8B in 2018.
A $330M industry worldwide, digital fitness trackers are currently the most popular type of wearable device.
Smart glasses, fitness bands, and watches are predicted to sell about 10 million units in 2014, generating $3B. By 2020, sales are predicted to surpass 100 million units.

1.2 BILLION
Smartphones are forecast to be shipped worldwide in 2016, as the smartphone market experiences rapid growth.

Source: FOCUSINFOGRAPHICS.COM
Book review

Penetration Testing – A Hands-On Introduction to Hacking

By Xavier Mertens

A few weeks ago I bought Georgia Weidman’s book about penetration testing: “A Hands-On Introduction to Hacking”. Being overloaded by many projects, I finally finished reading it and it’s now time to write a quick review. Georgia is an awesome person. There are not many recognized women in the information security landscape and Georgia is definitively one of them, I already met her a few times during security conferences! She started her own company, she’s a great speaker and the author of the SPF (“Smartphone Pentesting Framework”). That’s why I did not hesitate to buy her book.

The book title contains the word “Introduction” and, as explains Georgia in her introduction, this is the kind of book that you dream of when jumping into the penetration testing business. It covers indeed many topics but don’t be fooled by the title, it contains many tips and examples that could be useful also to experienced pentesters. Why? Sometimes people ask me how to “work in security” and I always compare information security to medicine. You have many specializations. It’s even more true for a pentester: web applications, reverse engineering, wireless, mobile devices, etc... It’s practically impossible to have a strong knowledge in all those ever-changing topics! That’s why Georgia’s book is a good reference. This is a technical book which focus on practical examples.

A first good surprise, the foreword was written by my friend Peter Van Eeckhoutte. If Peter accepted to put his name in a book, it’s a sign of quality! The book is big: 528 pages and 20 chapters which covers many topics. It is divided in five main sections which are organized like a regular pentest framework: from the reconnaissance phase up to the exploitation of found vulnerabilities:

- The basics
- Assessments
- Attacks
- Exploit development
- Mobile hacking

In the first section, “The basics”, Georgia covers some essential tools you need to master like the Kali Linux distribution or the Metasploit framework and how to deploy your own home lab. The “Assessments” explains how to collect information using open source intelligence, traffic capture and vulnerability scanners. The third section “Attacks” is the biggest and, of course, the most interesting funny! The following topics are reviewed:
Each chapter contains examples of commands, tips and some anecdotcs from Georgia’s previous engagements. It’s always good to learn from other’s experience. The next section is dedicated to exploit development (I suspect some contribution from Peter here ;-) ). Georgia explains how to exploit a stack-based buffer overflow in Linux and Windows environments. The chapter 19 is very interesting: it explains how to develop your own Metasploit modules. Finally, the last section covers mobile is based on Georgia’s own product the Smartphone Pentest Framework. At the end of the book a resources section lists all the references mentioned in the different chapters (link to tools, documentation, etc.). A goldmine!

While reading the book, I added plenty of bookmarks on pages that contains a specific command or useful command line switch. As said above, there are multiple domains to be pentested and we must keep us up-to-date. Everybody will learn by reading Georgia’s book! Keep in mind, the book focus on hacking and exploitation. No place for the homework like writing a report!

Xavier Mertens is an independent security consultant. His job focuses mainly on protecting his customer’s resources by applying "offensive" (pentesting) as well as "defensive" security (log management, SIEM, security visualisation). Instead of using out of the box solutions from security vendors, he prefers to advice on best ways to solve security issues. In parallel to his daily job, Xavier maintains his security blog (blog.rootshell.be), is a BruCON (www.brucon.org) co-organizer and offers some spare time and resources to initiatives like the EuroTrashSecurity (www.eurotrashsecurity.eu) podcast.

(This article was originally published on blog.rootshell.be/2014/07/30/book-review-penetration-testing-a-hands-on-introduction-to-hacking)

10 MOST COMMON ISSUES THAT CAUSE APPS TO GET REJECTED

(developer.apple.com)

1. Crashes and bugs
2. Broken links
3. Long load time
4. Linking to outside payment schemes
5. Localization glitches
6. Improper use of storage and filesystems
7. Crashes from users denying permissions
8. Improper use of icons and buttons
9. Misuse of trademarks and logos
10. Substandard user interface
A Field Guide To Mobile App Testing

Great mobile testing requires creativity to find problems that matter.

By Rosie Land

I’d like to take you through the thought process of testers and discuss the types of things they consider when testing a mobile app. The intention here is to highlight their thought processes and to show the coverage and depth that testers often go to.

Testers Ask Questions

At the heart of testing is the capability to ask challenging and relevant questions. You are on your way to becoming a good tester if you combine investigative and questioning skills with knowledge of technology and products.

For example, testers might ask:
- What platforms should this product work on?
- What is the app supposed to do?
- What happens if I do this?

Testers find questions in all sorts of places. It could be from conversations, designs, documentation, user feedback or the product itself. The options are huge... So, let’s dive in!

Where To Start Testing

In an ideal world, testers would all have up-to-date details on what is being built. In the real world, this is rare. So, like everyone else, testers make do with what they have. Don’t let this be an excuse not to test! Information used for testing can be gathered from many different sources, internally and externally.

At this stage questions, testers might ask these questions:
- What OS, platform and device should this app work on and be tested on?
- What kind of data is processed by the application (i.e. personal, credit cards, etc.)?
- Does the application integrate with external applications (APIs, data sources)?
- Does the app work with certain mobile browsers?
- What do existing customers say about the product?
- How much time is available for testing?
- What priorities and risks are there?
- Who is experiencing pain, and why?
- How are releases or updates made?

Based on the information gathered, testers can put together a plan on how to approach the testing. Budgets often determine how testing is approached. You would certainly approach testing differently if you had one day instead of a week or a month. Predicting outcomes gets much easier as you come to understand the team, its processes and the answers to many of these types of questions.

EXAMPLE: SOCIAL COMMENTARY ON THE FACEBOOK APP

I love using the Facebook app as an example when I’m gathering information as a tester. Complaints of it are everywhere. Just check out the comments in the iTunes App Store for some of the frustrations users are facing. Plenty more are dotted across the Web.
If I were challenged to test the Facebook app, I would definitely take this feedback into consideration. I would be daft not to!

**The Creativity Of Testers**

You probably know what the app is meant to do, but what can it do? And how will people actually use it? Testers are great at thinking outside of the box, trying out different things, asking “What if” and “Why” constantly. For example, mobile testers will often adopt the mindset of different types of people — not literally, of course, but the ability to think, analyze and visualize themselves as different users can be quite enlightening.

Testers might put themselves in these shoes:
- Novice user,
- Experienced user,
- Fan,
- Hacker,
- Competitor.

Many more personalities could be adopted; much of this really depends on what you are building. But it’s not just about personalities, but about behavior and workflows, too. People use products in strange ways. For example, they:
- Go back when they are not supposed to,
- Are impatient and hit keys multiple times,
- Enter incorrect data,
- Can’t figure out how to do something,
- Might not have the required setup,
- Might assume they know what they are doing (neglecting to read instructions, for example).

Testers look for these situations, often discovering unexpected results along the way. Sometimes the bugs initially found can appear small and insignificant, whereupon deeper investigation uncovers bigger problems.

Many of these issues can be identified up front with testing. When it comes to testing mobile apps, these might not all be relevant, but perhaps try asking questions such as these:
- Does it do what it says on the tin?
- Does the app perform the tasks it was designed to do?
- Does the app perform tasks that it wasn’t designed to do?
- How does the app perform when being used consistently or under a load? Is it sluggish?
- Does it crash? Does it update? Does it give feedback?
- Do crash reports give clues about the app?
- How can one navigate creatively, logically or negatively around the app?
- Does the user trust your brand?
- How secure is the user’s data?
- Is it possible to break or hack the app?
- What happens when you push the app to its limits?
- Does the app ask to turn on related services? (e.g. GPS, Wifi)? What if the user does? Or doesn’t?
- Where does the app redirect me? To the website? From website to app? Does it cause problems?
- Is communication and marketing consistent with the app’s function, design and content?
- What is the sign-up process like? Can it be done on the app? On a website?
- Does sign-up integrate with other services such as Facebook and Twitter?

**EXAMPLE: RUNKEEPER’S BUGGY UPDATE**

RunKeeper, an app to track your fitness activities, recently released an update with new “Goal Setting” features. I was interested in giving it a try, a bit from a testing perspective, but also as a genuinely interested user. I discovered a few problems.
It defaulted to pounds. I wanted weights in kilograms.

Switching between pounds and kilograms just didn’t work properly.

This ended up causing confusion and causing incorrect data and graphs to be shown when setting my goals.

Because of that, I wanted to delete the goals, but found there was no way to do it in the mobile app.

To work around this, I had to change my weight so that the app would register the goal as being completed.

I could then try adding the goal again.

Because of all of this confusion, I played around with it a bit more to see what other issues I could find.

Below are some screenshots of some of the issues found.

A recent update of RunKeeper included a new “Goals” section. Playing around with its dates, I discovered start and end dates could be set from the year 1 A.D. Also, why two years with “1”?

Another RunKeeper bug. This one is a typo in the “Current Weight” section. This happened when removing the data from the field. Typos are simple bugs to fix but look very unprofessional if ignored.

Here is the confusion that happened as a result of trying to switch between pounds and kilograms. If I want to lose 46 pounds, the bar actually shows 21 pounds.

There is no quick way to identify issues like these. Every app and team faces different challenges. However, one defining characteristic of testers is that they want to go beyond the limits, do the unusual, change things around, test over a long period of time — days, weeks or months instead of minutes — do what they have been told is not possible. These are the types of scenarios that often bring up bugs.

Where’s All The Data?

Testers like to have fun with data, sometimes to the frustration of developers. The reality is that confusing either the user or the software can be easy in the flow of information. This is ever more important with data- and cloud-based services; there is so much room for errors to occur.

Perhaps you could try checking out what happens in the following scenarios:

- The mobile device is full of data.
- The tester removes all of the data.
- The tester deletes the app. What happens to the data?
- The tester deletes then reinstalls the app.
- Too much or too little content causes the design or layout to change.
- Working with different times and time zones.
- Data does not sync.
- Syncing is interrupted.
- Data updates affect other services (such as websites and cloud services).
- Data is processed rapidly or in large amounts.
- Invalid data is used.
Testers like to test the limits of data, too. They will often get to know the app as a typical user would, but pushing the limits doesn’t take them long. Data is messy, and testers try to consider the types of users of the software and how to test in many different scenarios.

For example, they might try to do the following:
- Test the limits of user input,
  Play around with duplicate data,
- Test on brand new clean phone,
- Test on an old phone,
- Pre-populate the app with different types of data,
- Consider crowd-sourcing the testing,
- Automate some tests,
- Stress the app with some unexpected data to see how it copes,
- Analyze how information and data affects the user experience,
- Always question whether what they see is correct,

Creating Errors And Messages

I’m not here to talk about (good) error message design. Rather, I’m approaching this from a user and tester’s point of view. Errors and messages are such common places for testers to find problems.

QUESTIONS TO ASK ABOUT ERROR MESSAGES

Consider the following questions:
- Is the UI for errors acceptable?
- Are error messages accessible?
- Are error messages consistent?
- Are they helpful?
- Is the content appropriate?
- Do errors adhere to good practices and standards?
- Are the error messages security-conscious?
- Are logs and crashes accessible to user and developer?
- Have all errors been produced in testing?
- What state is the user left in after an error message?
- Have no errors appeared when they should have?

Error messages quite often creep into the user experience. Bad and unhelpful errors are everywhere. Trying to stop users from encountering error messages would be ideal, but this is probably impossible. Errors can be designed for and implemented and verified against expectations, but testers are great at finding unexpected bugs and at carefully considering whether what they see could be improved.

SOME EXAMPLES OF ERROR MESSAGES

I like the example below of an error message in the Facebook app on the iPhone. Not only is the text somewhat longwinded and sheepishly trying to cover many different scenarios, but there is also the possibility that the message gets lost into the ether.

Perhaps the messages below are candidates for the Hall of Fame of how not to write messages?
What about this one from The Guardian’s app for the iPad? What if I don’t want to “Retry”?

Platform-Specific Considerations

Becoming knowledgeable about the business, technology and design constraints of relevant platforms is crucial for any project team member.

So, what types of bugs do testers look for in mobile apps?

- Does it follow the design guidelines for that particular platform?
- How does the design compare with designs by competitors and in the industry?
- Does the product work with peripherals?
- Does the touchscreen support gestures (tap, double-tap, touch and hold, drag, shake, pinch, flick, swipe)?
- Is the app accessible?
- What happens when you change the orientation of the device?
- Does it make use of mapping and GPS?
- Is there a user guide?
- Is the email workflow user-friendly?
- Does the app work smoothly when sharing through social networks? Does it integrate with other social apps or websites?
- Does the app behave properly when the user is multitasking and switching between apps?
- Does the app update with a time stamp when the user pulls to refresh?
- What are the app’s default settings? Have they been adjusted?
- Does audio make a difference?

Connectivity Issues And Interruption

Funny things can happen when connections go up and down or you get interrupted unexpectedly.

Have you tried using the app in the following situations:

- Moving about?
- With Wi-Fi connectivity?
- Without Wi-Fi?
- On 3G?
- With intermittent connectivity?
- Set to airplane mode?
- When a phone call comes in?
- While receiving a text message?
- When receiving an app notification?
- With low or no battery life?
- When the app forces an update?
- When receiving a voicemail?

These types of tests are a breeding ground for errors and bugs. I highly recommend testing your app in these conditions — not just starting it up and checking to see that it works, but going through some user workflows and forcing connectivity and interruptions at particular intervals.
Does the app provide adequate feedback?
Does data get transmitted knowingly?
Does it grind to a halt and then crash?
What happens when the app is open?
What happens midway through a task?
Is it possible to lose your work?
Can you ignore a notification? What happens?
Can you respond to a notification? What happens?
Is any (error) messaging appropriate when something goes wrong?
What happens if your log-in expires or times out?

What happens if the OS is updated?
What happens if the OS is not updated?
Does the app automatically sync downloading to other devices via iTunes?
Is it worth automating some tasks or tests?
Does the app communicate with Web services? How would this make a difference?

Testing your mobile app after each release would be wise. Define a set of priority tests to cover at each new release, and make sure the tests are performed in a variety of conditions — perhaps on the most popular platforms. Over time, it might be worth automating some tests — but remember that automated tests are not a magic bullet; some problems are spotted only by a human eye.

Maintaining The App

Speeding up the process of testing an app is so easy. Test it once and it will be OK forever, right? Think again.

One problem I’m facing at the moment with some apps on my iPad is that they won’t download after being updated. As a user, this is very frustrating. Perhaps this is out of the control of the app’s developer. Who knows? All I know is that it doesn’t work for me as a user. I’ve tried removing the app and then reinstalling, but the problem still occurs. I’ve done a bit of searching; no luck with any of my questions, aside from suggestions to update my OS. Perhaps I’ll try that next… when I have time.

The point is, if the app was tested once and only once (or over a short period of time), many problems could have gone undetected. Your app might not have changed, but things all around it could make it break. When things are changing constantly and quickly, how does it affect your app? Ask yourself:

- Can I download the app?
- Can I download and install an update?
- Does the app still work after updating?
- Can I update the app when multiple updates are waiting?

Rosie has a mixed fascination related to software testing, the social web and startups. This includes running the Software Testing Club, The Ministry of Testing and a micro-Test Agency - Test Ninjas.

(Rosie has a mixed fascination related to software testing, the social web and startups. This includes running the Software Testing Club, The Ministry of Testing and a micro-Test Agency - Test Ninjas.)

Testing Is Not Clear-Cut

We’ve covered some ground of what mobile testing can cover, the basis of it being: with questions, we can find problems.

All too often, testing is thought of as being entirely logical, planned and predictable, full of processes, test scripts and test plans, passes and fails, green and red lights. This couldn’t be further from the truth. Sure, we can have these processes if and when necessary, but this shouldn’t be the result of what we do. We’re not here just to create test cases and find bugs. We’re here to find the problems that matter, to provide information of value that enables other project members to confidently decide when to release. And the best way we get there is by asking questions!

(This article was originally published on Smashing Magazine - http://www.smashingmagazine.com)
The Ins & Outs of Mobile App Testing

Removing the barriers to more and better mobile testing.

By Jim Cowart

Over the last decade, application testing has continually proved itself to be an important concern. When done well, testing can drastically reduce the number of bugs that make it into your release code (and thus actually affect your users). In addition, good testing approaches will help your team catch bugs earlier in the development lifecycle – resulting in a savings of both time and money (not to mention reputation with your users). Code that has good test coverage enables you and your team to make changes and introduce new features to your app without the fear of it breaking existing functionality.

The word “Testing” is a large umbrella, and is usually better understood when you break it down to specific types of testing. For example:

- Unit Testing – automated tests written by developers, with each test targeting a narrow slice of application behavior.
- Functional & Acceptance Testing – typically performed by QA personnel or automated UI testing frameworks.
- Performance Testing – often performed manually with profiling tools (for example – heap and CPU profiling tools), though many mobile app developers are moving towards integrating app analytics to gather this data from real usage as well.

That’s certainly not an exhaustive list of the types of testing, but enough to make an important point: mobile applications face several challenges when it comes to testing. Key among those challenges are:

- Platform & Device Diversity
- Immature Tooling
- Lack of Awareness

If you opt to write native applications for each target platform, then any code-level testing (i.e. – Unit Tests) will not be transportable as you move from Objective-C (iOS) to Java (Android). In addition, any scripted UI-Automation testing tools may not work for multiple platforms (or at the very least require separate scripts for each platform). Hybrid solutions like PhoneGap, or cross-compiled solutions like Xamarin can offer a single approach to unit testing (given a single codebase for multiple platforms) – but do not always offer the same level of quality as native tooling when it comes to performance profiling. Despite the trade-offs involved, I’ve found in my own experience that the biggest barrier to entry in mobile app testing is often a lack of awareness of what tools are available. That is the barrier which I hope to address in this post.

Unit Testing

Unit testing for specific platforms or cross-platform tools is not difficult, and your options abound. Let’s look at a sample of some of these choices.

iOS

iOS developers who’ve been writing Objective-C for a while may be familiar with OCUnit, which shipped with XCode prior to the XCTest framework. It’s still supported in XCode 5, but the understanding is that new and
future projects should focus on using XCTest.

Don’t let Apple’s sparse documentation on unit testing deter you from checking out the XCTest framework. If you’re running an OS X Server, you can also take advantage of the XCode service’s continuous integration features. As part of a continuous integration workflow, you can create “bots”, which can continually build and test your app.

Many developers prefer a Behavior-Driven-Development (BDD) style syntax for unit testing. If this describes you, be sure to check out Kiwi – a BDD style unit testing framework for iOS.

One other important mention is OCMock a mocking framework for iOS. Mocks are an indispensable part of writing adequate tests around your application’s behavior.

Android

JUnit is perhaps by far the most well known (and officially recommended by Google) testing framework for Android. The JUnit Android extensions allow you to mock Android components, but I’ve also seen quite a number of Android developers use JUnit with Mockito, another Android mocking framework.

Robolectric takes a different – and very interesting – approach by allowing you to run your Android unit tests in the normal JVM (Java Virtual Machine), without the need for an emulator. This enables your tests to not only run from within your IDE, but also as part of a continuous integration workflow.

Qt

Qt made the top 5 most used CPTs in 2013. If you’re building mobile applications with Qt, you’ll be happy to know about QTestLib, a unit testing framework built by Nokia. Based on my research, it appears that QTestLib can be integrated with a 3rd party continuous integration workflow – enabling very helpful testing automation.

PhoneGap/Apache Cordova

Web-based hybrid approaches to mobile apps can take advantage of a host of testing and mocking frameworks, not to mention scripted UI/acceptance testing tools as well (more on that in a moment). When it comes to unit testing JavaScript, three of the biggest names are QUnit, Mocha and Jasmine. I’ve personally used all three, with my favorite setup including Mocha and expect.js (which provides a BDD style test syntax). Mocking and “spy” frameworks abound in JavaScript as well, with Sinon.js and JsMockito among the more popular stand-alone mocking options.

Many PhoneGap developers take advantage of tools like PhantomJS – which is a “headless” (no UI) WebKit browser, with a JavaScript API. PhantomJS can be easily integrated into a continuous integration workflow to automatically run unit tests against your hybrid mobile application’s codebase.

Xamarin

Xamarin uses a customized version of NUnit (ported from JUnit), called NUnitLite which enables you to write unit tests against your Xamarin iOS & Android projects. For any shared codebase, you can use the unit testing framework of your choice.
Scripted UI Testing

Not every team can afford to hire an army of manual QA testers, despite how valuable that can be. Automated tooling can bridge the gap.

If you’re writing native iOS and Android apps, you’re in luck. Apple provides an “Automation instrument” that will automate UI tests against your iOS mobile application. The Android SDK provides the “uiautomator” library, described as “A Java library containing APIs to create customized functional UI tests, and an execution engine to automate and run the tests.” In addition to these, you can use third party tools.

Among the more interesting developments in mobile UI automated testing is the emergence of an open source project named Appium. Appium uses the WebDriver JsonWireProtocol to interact with iOS, Android and Firefox OS apps and gives you the choice of writing your UI tests in any WebDriver-compatible language (Java, Objective-C, JavaScript, PHP, Python, Ruby, C#, Clojure, Perl and others).

Performance & Profiling

Apple’s Instruments is one of the more impressive native toolsets I’ve seen recently. With Instruments, it’s possible to profile how your app executes, run stress tests, record and replay user actions, create custom instruments and a lot more. If you’re writing native iOS apps & not using Instruments, I recommend reading through the Quick Start to get up to speed.

With Android apps, you have several (albeit, lower-level) tools available: Systrace & Traceview. You can also use the Device Monitor to view memory usage based on logcat messages.

For hybrid mobile apps, you have a host of mature desktop browser tools (Chrome Developer Tools, Firefox/Firebug, etc.), which you can bring to bear on your app to profile CPU usage, memory, DOM manipulation and much more.

Many mobile developers have started taking advantage of third party analytics services such as Google Mobile Analytics, Countly, EQATEC, Flurry, Perfecto Mobile’s MobileCloud Monitoring and many others. The focus of these kind of analytics is usually more about how your app is actually used, user engagement, demographics, feature popularity, etc. However, it provides an opportunity to measure certain pieces of application performance from within real-world usage. While I wouldn’t recommend this being your first line-of-defense in performance testing, having the ability to track real world performance metrics can be a powerful tool in tuning your application to your users’ needs.

We’ve only scratched the surface of the various testing options available for mobile app development. What testing approaches & tools are you using when writing mobile apps? If you’re not currently testing your application, what are some factors that would change your mind?

Jim Cowart is a developer focusing these days on JavaScript, HTML, CSS, Node.js and open source in general. He’s most passionate about connecting with other developers – learning, sharing and promoting patterns and ideas that will make development more productive, enjoyable and empowering. Jim works at LeanKit as their Front End Architect and lives in Chattanooga with his wife and three sons. You can follow him on Twitter - @ifandelse

(This article was originally published on Developer Economics - www.DeveloperEconomics.com)
The Automation of Mobile Testing – Should we or Shouldn’t we . . .

Manual testing teams may not be able to test all the processes with each build.

By Sam Mullin

Test automation of applications has been around for many years. There are many of us in the automated testing field that started very early in the test automation phase, but the introduction of mobile devices has brought on a new angle in test automation of applications. Mobile device testing introduces new technical challenges along with new processes and testing tools. But should we use test automation for mobile applications that are on mobile devices? Many of us have mobile devices and we get our applications from the stores on Android or iOS and we go about our business.

Quality, performance and business rules?

As we know, there are at least three kinds of applications for Mobile Devices; Native, Mobile Web, and Hybrid. Like any application, developers need to build it, and testers test it. Now, when we introduce methodology into the mix, we add challenges. This agile development process introduces tight timelines and tight delivery dates. Developers are challenged to get the business processes into the applications as quickly as possible. This adds multiple builds daily, sometimes hourly. In the desktop and standard web testing process, whether automated or manual, we would expect to regression test each build, right? How many testers would you need to do that when builds are being delivered so quickly? Can manual testers do that and keep up with what they have tested, along with making sure that the business processes have been tested each time?

When there is a time crunch, usually there will be decisions made to stick to the critical business processes or new functionality and leave it at that. It sounds great, but is that the best way to test and make sure a quality product has been delivered to production? Users are accustomed to application pages being rendered within a few seconds, which means response times need to be verified before going into production. Users are accustomed to the application functionality being consistent each and every build. If there is a new build, users are not expecting to lose functionality or any visual effects. Our eyes are accustomed to application pages looking great each and every build. How is a small test team going to do that? Honestly, a manual testing team will probably not be able to test all the processes each time and will revert to the 80/20 rule, or something close. Now, there are some teams out there that will claim to have an army of testers that they will call up and have the start testing the build. That is awesome, however, even that army of testers may take the common path or the happy path. They may not have all the business rules or the defects to look at to see what the challenges or critical areas of the app are. How do they report back to a central point and collaborate on what they have tested and what the results were? How would they also add requirements trace-ability into the process? Unless this is a seasoned group, that will be a challenge.

Where test automation comes in.

From wherever you are in the world, using test automa-
tion can make testing more complete and add require-
ments coverage, test coverage, trace-ability, defect
tracking, and overall test reporting easier to manage.
From one report, management can see what percent-
age of the application has been tested, what defects
have been introduced and what defects have been
fixed. With test automation, when a defect is found, it
is very easy to recreate that defect and the steps for it.
When manual testers test and find a defect, if the
steps are complex enough, then chances are, the
tester will not follow the same path and the defect may
not be reproduced. That may be a big issue.

With test automation, test scripts can be run off hours
if needed. Now with mobile device testing, that will add
complexity because some environments and devices
will need to be configured to run the test. That may or
may not be able to be automated. Either way, the
scripts can be run and tests can be run without
manual intervention once set up. With test automa-
tion, when builds are done multiple times during the
day, we can find the changes to the application much
faster than a manual tester.

How fast can scripts do that?

Well, that would depend upon the environments being
used, the complexity of the application under test, and
the performance of the application.

For the most part, running automated scripts will
always test more functionality than a manual tester. If
there is a critical path that needs to be tested, the
scripts can be run specifically for that and they will be
executed the same each and every time. For many
teams, cost comes into the picture. Most departments
have budgets and test automation tools are usually
pretty expensive. Once research is done, it will be
found that there are many tools out there that can
create scripts for a fairly inexpensive cost with even
cheaper renewal costs. For the teams that do not have
much experience with test automation, it would be
better to try a small project with one of the less expen-
sive tools. And remember, cheaper cost does not
mean cheaper in the way the tool works, but it may
mean that you don’t get all the bells and whistles.
Compromises will always need to be added.

Should you use test automation for mobile device
testing?

We can ask the same question for the test automation
of a desktop application. It does not just need to be
answered for mobile device testing. If your team plans
to create subsequent updates to the app, or create the
application in phases, I would highly recommend some
sort of test script automation. If the plan is to just
create a quick application one time and that is all, then
it is up to the team to decide whether or not it is cost
effective to automate the testing process. There will
always be a benefit to test automation as a whole
because test automation is not just script execution.
Test automation is the whole testing process. That is
where the real benefit for the whole project comes in.
But for development and testers, script automation will
help by allowing the manual testers to move onto
some other critical tasks and leave the regression to
the automation team.

Whether you have 4 or 400 testers, there will always
be something that will be missed. Adding test automa-
tion will add test coverage and higher requirements
coverage to your testing results. Mobile Device test
automation will use a lot of the same processes that
standard test automation uses. The tools may be
different, but the rest will be similar.

Sam Mullin is a specialist in
Mobile Device Test Automa-
tion, he has contributed to
test planning, design, and
execution of automated
testing efforts within the
healthcare, electricity, and
banking industries. He continues to work on
providing specialized support for mobile device test
automation using various mobile device testing
tools. Currently, Sam is working for Olenick and
Associates located in Chicago Illinois (www.ole-
nick.com)

(This article was originally published on www.ole-
nick.com/blog/automation-mobile-testing/)
Mobile application testing is taking on greater significance as just about everything done on the desktop is transitioning to mobile. It’s taken a while for mobile testing in the consumer space to be made much of a priority. Now that sensitive data is being passed back and forth between mobile devices, apps and appliances, it’s imperative that the same quality and coverage of testing is done on mobile devices as is routinely done for desktop applications.

The mobile ecosystem introduces quite a bit of complexity. Tests must be executed on multiple devices and multiple operating systems, and often across multiple networks. And, it has to be done on a frequent basis due to the relatively rapid introduction of new devices and faster/better operating systems. There are a number of ways to do large-scale mobile testing, but not all provide the same level of control. To have the utmost confidence in your mobile apps, in-house testing of known devices is essential, and test automation can make it manageable.

The primary factor that determines an automation tool’s success is the ability to work across platforms and mobile technologies. For most testers this means simultaneous testing of Android and iOS devices.

TestArchitect Mobile Plus offers testers a viable mobile testing platform for Android and iOS. Since the introduction of the mobile testing edition last year, the tool has undergone quite a bit of refinement. With the latest release, TestArchitect 8, it supports the latest Android version, as well as iOS 7. This platform support combination provides testers with the ability to test on devices that make up the vast majority of those used in the market.

For Android there is support for developed SDK and WebView apps or native apps. Additionally, the APK file can easily be resigned, eliminating the need for the mobile device to be jail-broken. Tight control of iOS by Apple tends to limit just how much tool developers can access, TestArchitect provides testing of built-in and hybrid apps. But it is the testing functionality is what sets it apart from other mobile testing tools.
Test creation is simplified with object-based and native method testing, and an extensive built-in library of system-level actions for multi-touch gesture and device access. By installing a small communication agent on the phone or tablet device, test cases are executed on the client where the non-modified application running on the mobile device can be accessed exactly like a real-life scenario.

The additional functionality of the Keypoint algorithm detection provides sophisticated image recognition and makes it possible to automating testing for fast-moving graphics and gaming applications. The check picture feature allows setting optimum thresholds for images comparisons expected during automated testing. Debugging is enhanced with screenshot recording of the UI display during test execution.

The challenge for all mobile testers is the sheer number of devices that have to be tested. TestArchitect goes a long way to help testers in this area. The tool provides simultaneous, multi-device test execution and monitoring through USB or Wi-Fi connection to a virtual emulator or to the physical mobile device. The ability to test more than 40+ devices connected to the client machine provides testers with tight control over the testing on devices of known quality, providing the highest quality assurance. By going this route, critical functionality and performance can be tested on a good representation of the most popular devices in the market.

What really sets TestArchitect Mobile Plus apart from other tools is that it’s an integrated test automaton platform that combines test methodology, automation technology and test management. The platform is based on the Action-Based Test (ABT) method to efficiently create, automate and manage tests. ABT enables design-centric test creation and execution, without scripting or programming. Readable action keywords allow even novice testers to quickly ramp up testing and create a set of manageable automated tests that the entire test team can read, understand and share—and reuse with only minimal updating of actions when the application changes.

Mobile testing will continue to grow in importance, and the need for automation will only increase. TestArchitect Mobile Plus is a good choice to consider for testers that are looking for a solution for their mobile testing challenges.
**Glossary: Mobile Testing**

**Mobility**

While mobile usually refers to mobile device, mobile apps, mobile platform. Mobility is a term describing not only the device but access for employees accessing corporate data from any location, cloud storage and cloud API services, mobile context awareness, integration with a variety of Internet of Things devices. It’s access to the data, products, and services enabled by being mobile.

**Context-Aware**

Context-aware mobility provides the ability to dynamically capture and use contextual information about mobile assets to optimize, change, or create communications flow and business processes. Contextual information can be collected for any mobile asset involved in a business process, and this includes not just devices and products but also people. For instance, a mobile asset can be a worker, a customer, or a patient, or it can be a pallet of finished goods. (Source: Cisco)

Context Awareness provides intelligence or information based on context. Mobility offers the opportunity to gain awareness of the individual and their interactions. That means location, biometrics, weather data, data about other individuals, and any other relative data based on mobile context will be used to deliver a fuller environmental awareness. (Source: Frank Yue, LTE World Series)

**Wireless technology communication protocols**

Test Engineers need to be familiar with various parts of the Internet of things and mobility. Comprised of sensors, embedded systems, connectivity and remote control by various communication protocols, these communications differ in function, network infrastructure, power consumption, and range.

- Wi-Fi
- Bluetooth, and its varieties, (Bluetooth low energy, ANT, ZigBee, etc)
- RFID and NFC
- GPS
- Cellular - 3G, 4G LTE, WiMAX

**Wearable**

Wearable computers, also known as body-borne computers or wearables are miniature electronic devices that are worn by the bearer under, with or on top of clothing. This class of wearable technology has been developed for general or special purpose information technologies and media development. Wearable computers are especially useful for applications that require more complex computational support than just hardware coded logics. (Source: Wikipedia)

**Remote control**

Remote control is a component of an electronics device—most commonly a television set, DVD player, or home theater system—originally used for operating the device wirelessly from a short line-of-sight distance. Remote control has continually evolved and advanced over recent years to include the internet, Bluetooth connectivity, motion sensor-enabled capabilities and voice control (Source: Wikipedia)

**Proximity sensor**

A proximity sensor is able to detect the presence of nearby objects without any physical contact. A proximity sensor often emits an electromagnetic field or a beam of electromagnetic radiation (infrared, for instance), and looks for changes in the field or return signal (Source: Wikipedia)

**NFC**

NFC, Near Field Communication, is a short-range wireless technology used as a method of transferring data over short distances between two compatible devices. Now commonly used for retail transaction payment (Source: Wikipedia)

**RFID**

Radio-Frequency Identification (RFID) is the use of radio waves to read and capture information stored on a tag attached to an object. A tag can be read from up to several feet away and does not need to be within direct line-of-sight of the reader to be tracked. RFID technology is used in smart cards, tracking-for dogs, livestock and everything! RFID technology is everywhere. It’s the duct tape that holds the digital world together. (Source: www.epc-rfid.info/rfid)