Going Mobile: Testing Beyond the Device

By Jonathan Kohl

Originally Published by The Testing Planet, March 2012

Have you noticed a shift in technology lately? If you haven't, take a look around you next time you go to a restaurant. You will probably see people using their smart phones or tablets: fingers tapping away, eyes staring at the screen, an eerie blue lighting up their face. They are focused, engrossed, addicted, and not to the food, company and atmosphere in the restaurant. They are interacting with (and addicted to) applications on mobile devices that are convenient to carry with us wherever we go.

Mobile devices are our contact points to applications, services, the internet, and increasingly, our social lives. For example, consider that trip to a local restaurant and how mobile technology has become enmeshed in that experience. How many of the following activities can you relate to?

- Searching for a restaurant nearby (using location-based services or GPS.)
- Selecting a restaurant based on cuisine, location or price.
- Reading user reviews and ratings to help make a decision.
- Plotting out and following directions to the restaurant on a map.
- After arriving, checking in to a social networking application, alerting people in your network of your current location.
- Searching the web to answer questions about the items on the menu. (I always forget what aioli is.)
- Translating a meal that is in a foreign language.
- Checking the nutritional information for a meal.
- Once the food arrives, taking a picture of it and uploading it to your social networking profile.
- Friends commenting about the photo and your meal.
- Throughout the meal, posting and responding to comments about the meal on your social networks.
- At the conclusion of a meal, posting positive, neutral or negative comments about the experience to restaurant review applications.
- If the experience was poor, ranting on public social media.

We haven't even touched on the buzzing and beeping that occurs while receiving emails, text messages, and phone calls. Just imagine the technology that is required to support all of these activities. If you dissect it, the infrastructure and underlying complexity is amazing. The mobile device is just our primary contact point to a large amount of technology that supports our mobile-enhanced, social and work activities. All those photos we upload and share, all that data we have access to and send to our social networks, all have to be stored somewhere and quickly accessible not only to us, but to those in our networks. This provides new challenges from a testing perspective.

The people who are using these applications are using them on the move. They interact with them physically, in different environments with different underlying infrastructures and in all weather. As testers, we need to interact with the application, find things that are difficult to input into, and to see, and find problems that are caused by being in interesting locations. These are the problems your end users will encounter the most, and will frustrate them the most. (I have developed a framework to help with testing these kinds of activities called I SLICED UP FUN!.) However, merely testing the front end of many mobile applications ignores the support infrastructure underneath that allows us to work with the devices the way we do. We also need to analyze that and figure out potential weak points that we can exploit when testing. Two areas that have testing implications in the mobile space are data and the cloud.

Most testers are familiar with data storage, and software interactions with data-related software and hardware. If you aren't familiar with "the cloud", they are machines that are available for use for different purposes for a fee. You essentially rent time, space and processing power from a company that provides you with access to their machines over the internet. It's a relatively inexpensive way to add processing power, distribute access and store and distribute data. There are also GPU (graphics processing unit) cloud centers that can be used for graphics rendering, and movement in areas like grid computing to help distribute processing across a wide array of machines.

Data storage and hardware/software/network interactions can have an impact on performance. Using the cloud is a powerful tool, but I've had the following challenges when testing software on the cloud:

- You lose the complete control you have over your own equipment.
- Other companies and projects are also utilizing machines from that cloud service, and their software can potentially slow everyone else down if they are doing something that taxes the network capabilities of that particular data center.
- Cloud services may not have the uptime you are used to
- There are service outages, and bugs that cause longer, unplanned outages.

These issues can cause time and frustration when trying to track down performance issues or timing-related bugs.

Here's an example: a team I was working with had developed a mobile application on several popular smart phone and tablet platforms. They already had a web infrastructure in place for a web-based application, but they exploited mobile devices to help put media in people's hands. They were able to use features from mobile devices to enhance this experience, and provide connectivity and productivity for remote workers who were often away from desktops or laptops. The back end infrastructure was a typical web application using the LAMP stack (Linux OS, Apache web server, MySQL database and PHP as the programming language). This worked well for web applications, but they weren't sure how the extra load of more and more people connecting and downloading and uploading media files would stress the system. I joined the project team to help with performance and functional testing.

We tried using load generation tools to simulate the extra web traffic, but this didn't address the media downloading and interactions on mobile devices. To simulate this effect, we used functional test automation tools to load the application in a web browser, sized to the same size as the mobile screen, and simulated user input by sending messages directly at the protocol level. We then virtual servers and an internal cloud infrastructure to generate load across multiple machines. After jumping through a few technical hurdles, we were able to remotely execute tests on several machines on different networks, with about eight simulated mobile clients on each machine. This effort revealed some astounding results. The server behaved quite differently with the mobile client than with the old web client, and some of the simulated mobile clients suffered from serious performance and functional problems.

Once these issues had been sorted out , the process was reversed . Now that we knew some of the common trouble spots, and were able to simulate those conditions on the server, while testing with a small number of mobile devices to see how the real thing handled server performance problems.

After we felt confident that we had dealt with most of the performance problems using both approaches, we pushed out builds to our beta testers and had them re-test the application in the field. Some of them still reported serious performance issues, and a pattern emerged: the people who were the furthest away from our data center had the worst performance.

Data Implication: Distributed Data with the Cloud

We discovered that when using large media files, people who were furthest away from the data center experienced the poorest performance. It turned out that the combination of lower processing power on the mobile devices compared to desktop computers, coupled with the complexity and differing speeds of data networks made performance problems much more apparent. Performance issues you barely noticed on a desktop were unbearable on a mobile device.

We needed a quick fix, so we looked at what we had done to design tests: we decided to use the cloud. We were already using cloud services for some of our media storage, so we expanded those services to data centers that were closer to the geographic areas where users experienced poor performance. This data distribution using different cloud centers worked well. This was a bit of a shock for us, but if you model the system from storage to download and interaction, it begins to make sense.

Data Implication: Reduce Bottlenecks with noSQL

Social networking infrastructures, and systems with high volume over a large geographic area, have highlighted interesting data interaction bottlenecks. If you look at a typical web application such as a LAMP stack, you tend to have a synchronous flow to and from the database, and most of us use some sort of Entity-Relationship Model and SQL to access the data. (Even if the SQL is generated by an Object-Relational Mapper.) When you have many reads and writes with a large amount of data, with different circles of people who require instant, simultaneous access, you inevitably run into bottleneck issues, which translate into poor performance at the end user level. Man social applications store an enormous amount of data, with many users posting pictures, video, sound files and other items in their systems. Add more users with the ease of use that mobile devices provide, and bottleneck and performance problems become worse.

The noSQL is a movement that gained popularity, in part to help deal with some of these problems. Social networking systems have different data usage and requirements than other applications, so as programmers tried to solve these problems by adding parallel database interactions, different hardware and software solutions, they inevitably began to challenge the Entity-Relationship Data (ERD) model itself. What many found is that the ERD model we know and love does not always work for social data with its various relationships, and constant interaction. Without boring you with technical details, noSQL is a different model to save, access, update and read data than a typical database. There are different technologies used, many of them are also built to scale rapidly and cope with the rapid growth

and massive data requirements that social and similar systems depend on.
From a testing perspective, this means we have to change the way we interact with data, and challenge some of our assumptions about how it is stored. noSQL systems may use something that looks like a database, or they may store data in different sorts of files. Querying may use something altogether different than SQL. One of my favorite SQL alternatives is XQuery's FLWOR (flower) expressions. We also have to look at configuration and performance, with less testing tool support than what we enjoy with traditional, ERD systems.

Many noSQL tools are newly developed, but the storage principles behind them are the same as any other persistence model. You will need to adjust how you think about storage, access and learn different tools.

Bringing it All Together

In many cases, the application we interact with on our mobile device represents the thin end of the wedge, or the tip of the technology iceberg. If we run our applications in ideal environments under ideal conditions, and ignore the infrastructure they depend on, we will miss important bugs. The problems our end users run into aren't necessarily that easy to discover if we are sitting in our test lab under ideal conditions. What happens to the winter user who tries to use your device with gloves on? What problems might someone in a different country encounter? What does our back end infrastructure look like? Does it depend on new technology? How might it perform and handle data uploads and downloads seamlessly? What complications do the size of and type of data our users require our system to handle? How might their physical location in relation to our data center have an impact? As mobile testers, we need to look at the entire system and come up with creative ways to simulate real-world conditions using different kinds of tools and techniques to help discover the problems that are important to our users. In this article, we've looked at two specific areas: data and the cloud, but there are more. When testing mobile devices, research and look beyond the device to get the full picture. Only testing from the application-level may not be enough. Testing mobile applications with the full picture in mind will lead you to more of the important problems that your users are likely to encounter.

Author Bio

Jonathan Kohl is an internationally recognized consultant and technical leader. Based in Calgary, Alberta, Canada he is the founder and principal software consultant of Kohl Concepts, Inc. Jonathan helps companies define and implement their ideas into products, coaches practitioners as they develop software on teams, and works with leaders helping them define and implement their strategic vision. He is also a popular author and speaker. As a thought leader in mobile application testing, exploratory testing, developing policy and strategy, and helping teams adjust to methodology changes, Jonathan doesn't just write and talk about developing software, he actively helps teams deliver the best products they can.

[1] - http://www.kohl.ca/articles/ISLICEDUPFUN.pdf

[2] - http://radar.oreilly.com/2012/02/nosql-non-relational-database.html

[3] - http://www.strozzi.it/cgi-bin/CSA/tw7/I/en_US/nosql/Home%20Page

[4] - http://www.w3schools.com/xquery/xquery_flwor.asp