testmunk

Articles

Guest Blog: Flipboard transitioning from manual to automated UI testing

Posted by Derrick Lam - Flipboard QA on April 21st, 2015

Here at Testmunk, we are fortunate to work with exciting clients, and with some of the coolest apps (and app developers) out there. Today we'd like to give the floor to Derrick Lam, Android Mobile QA at Flipboard. He'll be providing a high-level overview of some of his experiences with introducing automated functional UI testing to their QA process, and more specifically, about some of the bugs it has helped to find. -Martin

Derrick Lam

Thanks. As Martin stated, I'm Derrick Lam, QA for Android at Flipboard.

Flipboard is a newsreader app that aggregates content from all over the world and presents it to the user based on his or her interests. It has a simple, intuitive interface, allowing you to configure your interests on the fly, and offers the top news of the day based entirely on your choices. By bringing the content you want, and only the content you want, Flipboard has achieved over 100 million downloads and is one of the top downloaded iOS and Android Apps. We have been featured by Apple and the Play Store more than once and have gained a strong reputation for excellence. As you can imagine,

this sort of attention means that we have to be highly cognizant of the quality of our product and be sure we never release a feature that isn't ready.

As you know, building an industry-leading Android or iOS app requires a great deal of testing. Every release has its own challenges. To ensure success, one has to imagine nearly every use case possible and consider all the devices that your app might open on. Until a few months ago, our QA and regression testing were a highly manual process. It was obvious to us that to keep up manually and still manage to cover all of the required regression testing would soon either limit what could be accomplished in a build cycle or force us to lower our standards. Neither option appealed to us in the slightest. For this reason, besides automated unit testing, we began to use automated functional UI testing.

Functional UI tests have improved our product and allowed us to test more thoroughly and accomplish more in a build cycle than ever before. With the help of the Testmunk team, we were able to quickly set up over 30 testcases, covering a multitude of scenarios. We started to run these testcases several times per day on circa 25 different phones. The execution of these testruns is performed in parallel on all the devices and take around 1.5 minutes per testcase. In order to perform these same testcases manually would have taken us several hours.

As an audience of avid developers and iOS and Android enthusiasts, I thought that you might be interested in some of the bugs that functional UI testing has been able to find and report. Quite frankly, some would also have been found manually. However, others might never have been found within the limited time-frame of a build cycle and could easily have been released for the public to discover. Because of these favorable results, we are currently working to build out more functional UI tests. It has been great to see how automation has not only provided us additional confidence and a safety net, but at the same time has helped to launch our versions in the fastest way possible. Long story short, if our goal is for our users to have the best news-reader/personal magazine app possible, this means catching bugs before they are being released.

Automated UI testing helped us find numerous bugs during the course of a development

cycle. We ran our testsuite on individual devices here and there as needed, but the biggest automation value lay in the ability to execute on over 30 different Android devices at once over Testmunk. We were also able to configure Jenkins so that with every code push to the master branch, the testcases would be automatically executed on Testmunk's devices against the latest app built. Some of the bugs found by automated UI testing in the past months typically fall into the below categories:

- Backend issues
- UI Visual Issues
- Crashes
- Other Issues

Identifying Backend Issues

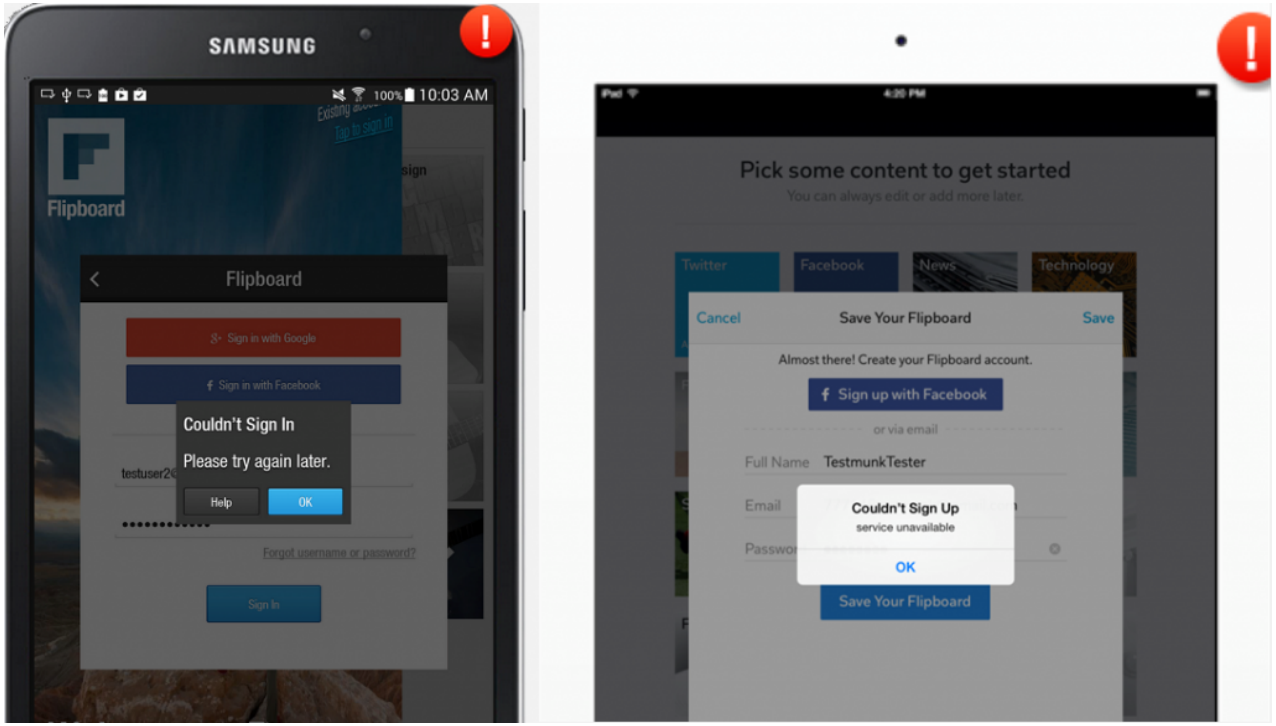
During the course of executing our testsuite, we were able to identify and report several bugs to our back-end engineers. This was an unexpected bonus, because, quite frankly, we tend to work more closely with the mobile developers first. Back-end issues are difficult to predict, as they represent an external factor with a potentially big influence over how the front-end UI behaves. The app may fail in the end even with strong unit and integration tests in place. Without a proper testing environment, a QA engineer may not be able to simulate the conditions that will lead to the discovery of flawed connections.

One of the features that helped greatly in determining back-end issues was the ability to set
"time-out" parameters for various steps. This helps in determining not only whether the app and back-end can connect, but whether they do so in a timely fashion. This provides valuable insight into what the actual user experience is like. Coupled with the ability to download logs directly from a failed test, and screenshots laid out for all devices arranged by teststep, we are able to catch more than just UI specific issues. Functional tests allow us to detect several issues such as:

- backend servers that are not responding in a timely manner, resulting in:
    - Login issues
    - inability to retrieve content
    - content not retrieved in a set time, causing errors
    - image or frames not displaying correctly (user interface might be affected)

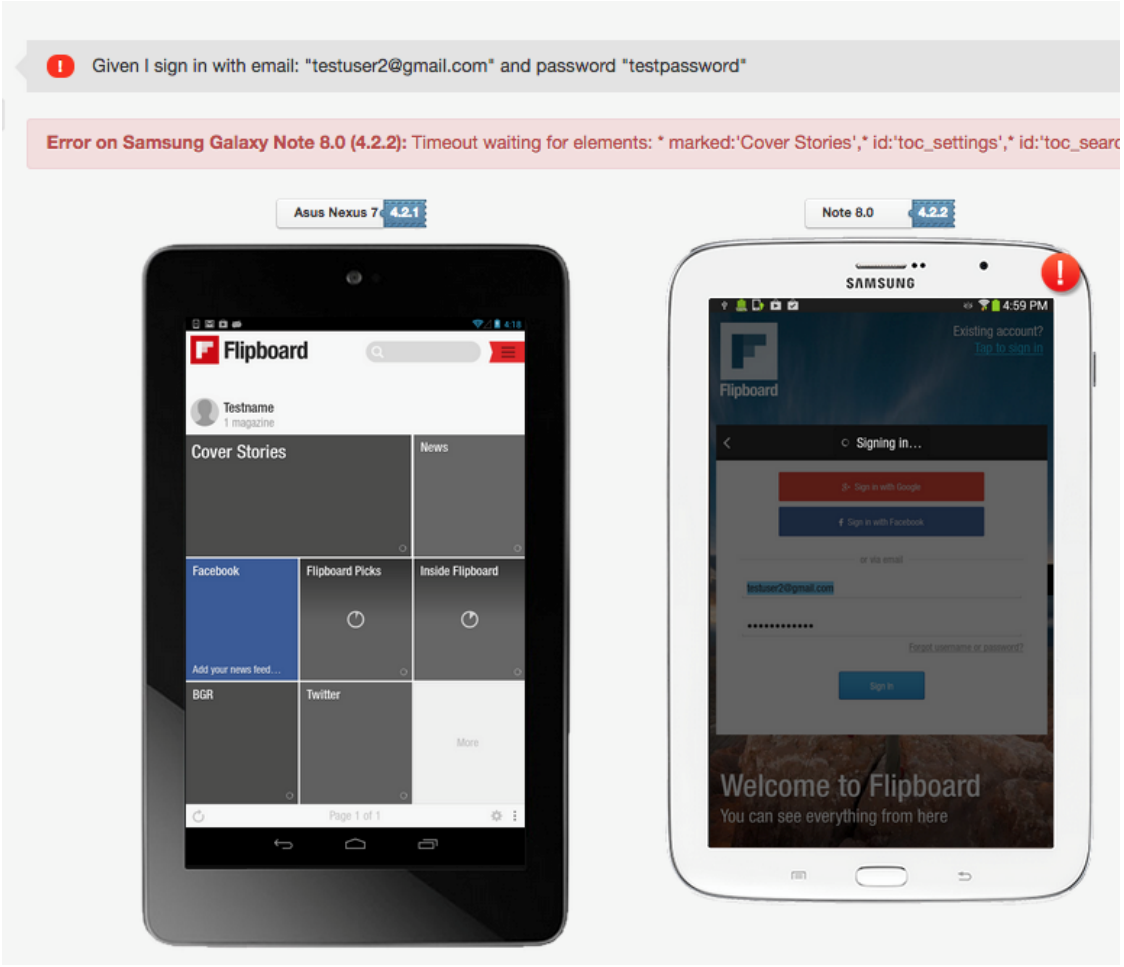
Login issues

Here we see a problem with login, identified as an issue with the backend server. In a few cases, the front-end team could not identify the particular cause of the issue, however, reporting the problem to the back-end team allowed them to investigate the matter, and identify the problem. In this case, unit or integration tests might not catch the issue because technically, communication between device and server was happening. Some devices passed, and on others, signing in took more than 30 seconds. This means that the app and the server both knew what should be done and could accomplish the task. However, a successful login that takes too long is not a successful test.



Back-end Sign In Issues

Setting a default time-out rule to each teststep ensures that none of our typical user interactions will experience lag or cause application errors, and that our servers are able to properly handle all back-end requests. Testmunk not only allowed us to establish default timeout rules, but also to configure per step timeout rules (either more rigorous or more lax, depending on the teststep) as needed for our product. By configuring the test-scripts, we were also able to log all actions and the time taken.



Time Out Issue

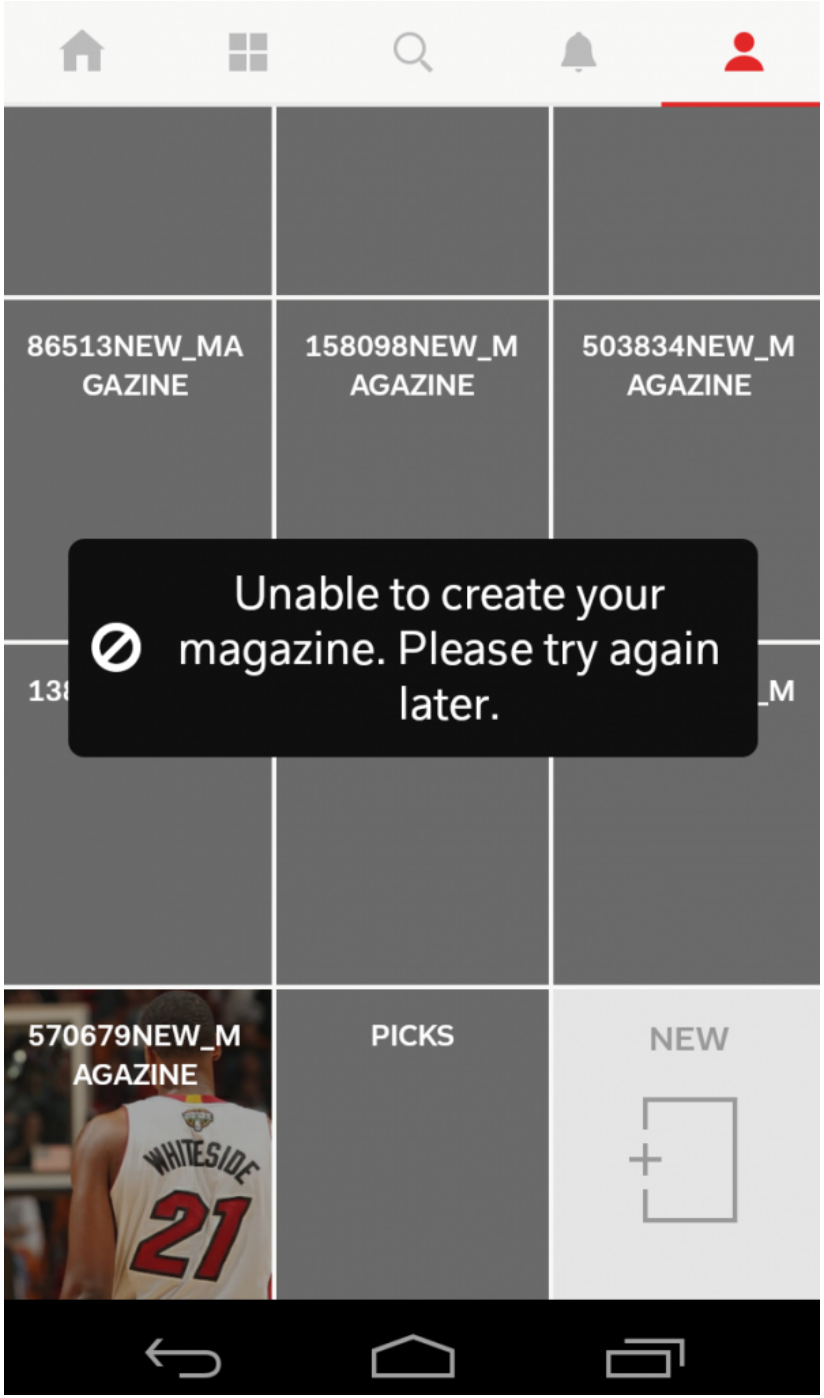
Problems Retrieving Content

Functional UI testing can not only determine whether responses are happening in a timely manner, but whether they are retrieving what they should at all. This example shows a WebView with an error code. Without side by side, correlated results, this issue might be considered an isolated incident and not receive the attention it is due, until a second or third device comes up with the same problem. When it crops up on several OS versions and device models, (within the same short testrun) it not only gets the attention

it deserves, but may offer a clue as to the cause, by examining the commonalities between the devices.

Bad Gateway Issue

In the above case, the same issue retrieving WebView data temporarily prevented users from creating their own magazines in the app. As I'm sure you realize, this is a rather important feature of the app (massive understatement). Automated testing helped to link these two issues quickly, simply by allowing them to be discovered in the same testrun.



Magazine Creation Issue (Related to Bad Gateway)

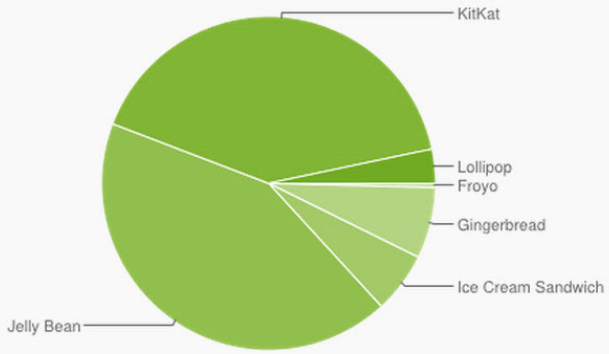
Verifying UI issues

User interface issues, in particular those specific to certain OS versions or devices, are far harder to catch with manual testing, simply because manual testers often don't have a large enough collection of devices to test on. Even if an issue is found on the first few android devices, it cannot be assumed that the problem is pervasive (affecting all android devices). If you assume a manual test done on the first 5 of 10 devices, and you have them organized by OS version, it might only tell you that the issue affects versions prior to 4.0. An assumption that it affects all versions might potentially lead you in entirely the wrong direction.

Device fragmentation, especially on android (though they are getting much better, as shown below), can make it difficult for developers to identify UI glitches during development, simply because each tester may not have access to a full range of devices across the relevant OS platforms. Having access to a large set of devices, such as Testmunk provides, and being able to kick off a detailed testrun on all of them simultaneously is a huge help.

Version	Codename	API	Distribution
2.2	Froyo	8	0.4%
2.3.3 - 2.3.7	Gingerbread	10	6.9%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	5.9%
4.1.x	Jelly Bean	16	17.3%
4.2.x		17	19.4%
4.3		18	5.9%
4.4	KitKat	19	40.9%
5.0	Lollipop	21	3.3%

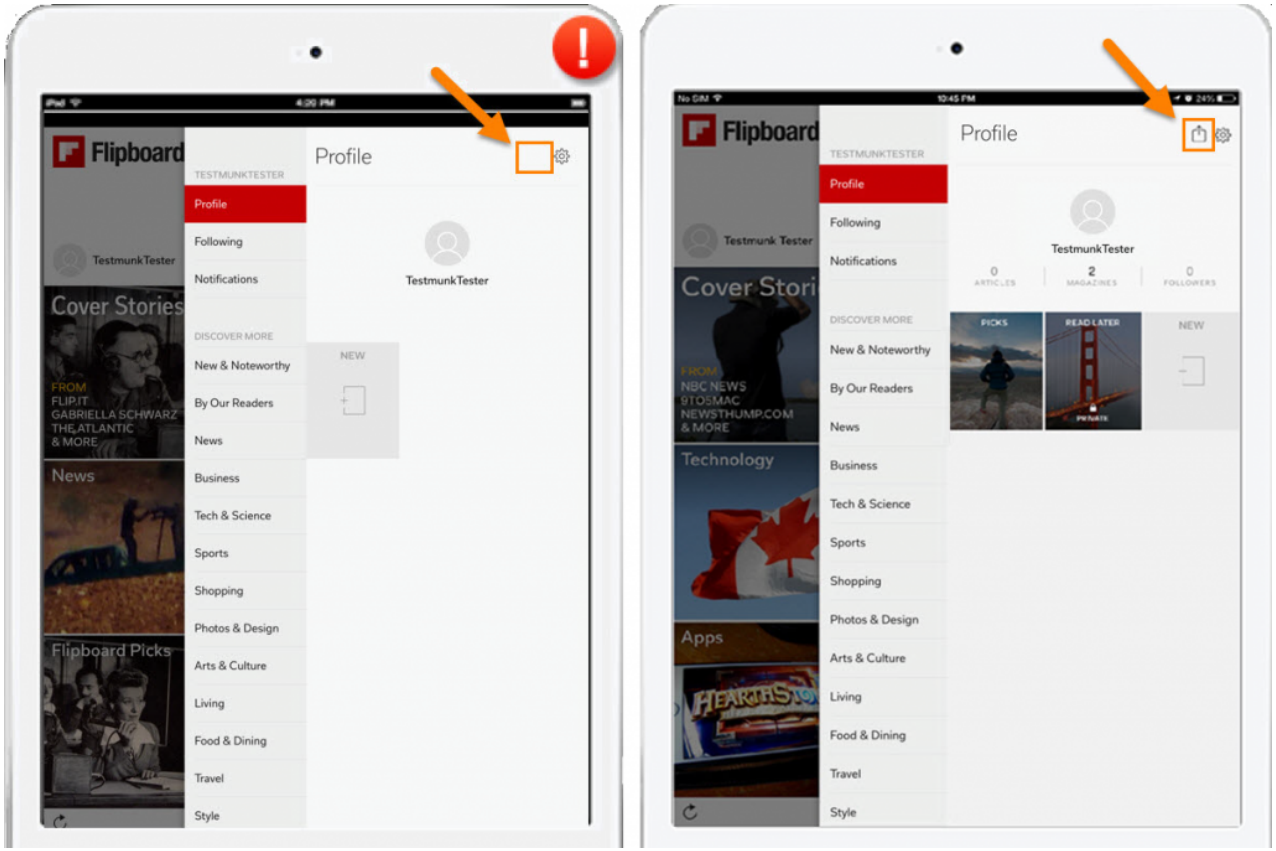
Data collected during a 7-day period ending on March 2, 2015.
Any versions with less than 0.1% distribution are not shown.



Source:
https://developer.android.com/about/dashboards/index.html?utm_source=suzunone

Android Developer Dashboard on Fragmentation

Take, for example the error seen below, where in one of regression tests the share option in the top right corner was inexplicably missing. In addition, selected magazines were not represented. This type of issue is very hard to detect if only present on a handful of devices, especially if a full range of devices and OSes are not available in the environment. A single missed OS or device version can result in a missed bug.

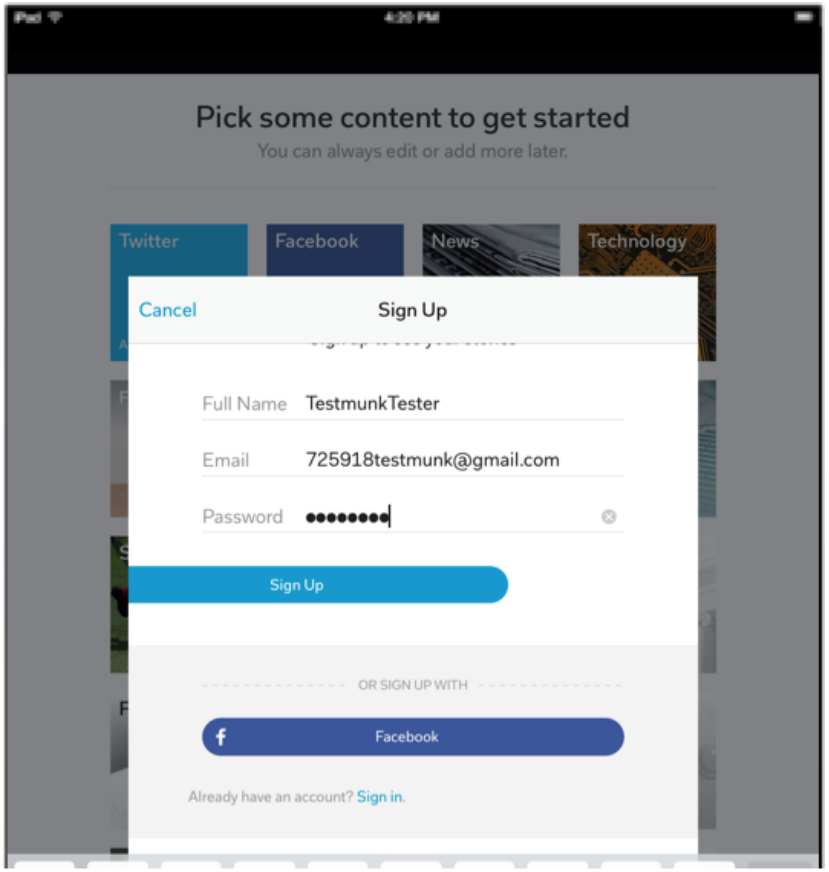


Share Tab Missing

The time taken to do this minimal amount of testing (5 devices) manually (by one person) is roughly equivalent to a full testrun across 25 plus devices, with several other testcases included. And the best part is, you don't have to make assumptions. You can see that the issue is indeed across all iOS devices, or only Android devices of Kit Kat and above. The extent of an issue and the devices it affects is crucial to troubleshooting, and if you don't have enough devices to test the whole list, or simply don't have the time to test them all,

you may not get the big picture you need to prevent a bug from squeaking through.

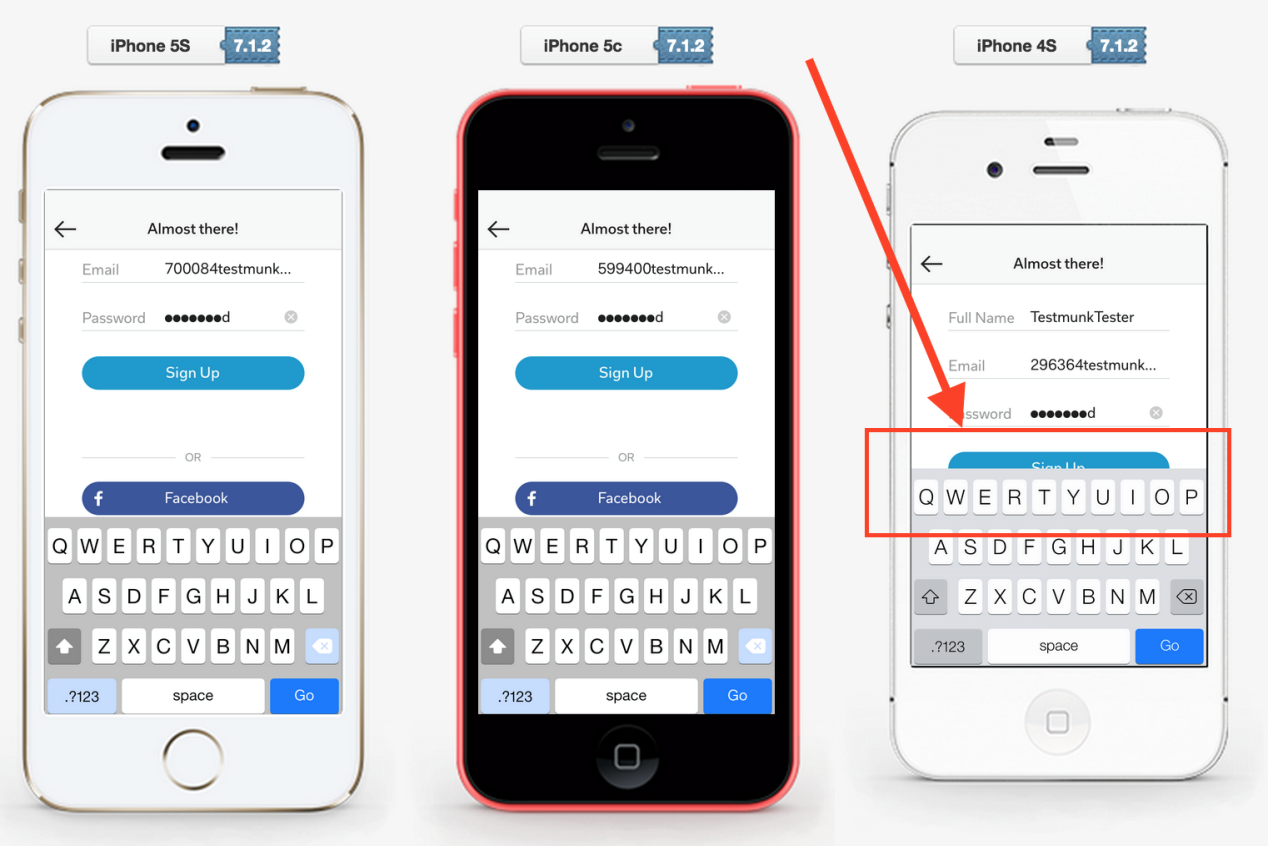
In order to test the same UI screen on each Apple device, across each of the older devices and OS versions, then again across the far larger number of Android variants can be a tester's nightmare. An issue such as the one below, which only occurred on iOS 8 with retina display, might not have been discovered without Testmunk's automated platform that provides screenshots of every step, and lines up the results by teststep, for easy comparison.



Sign Up Button Not Centered

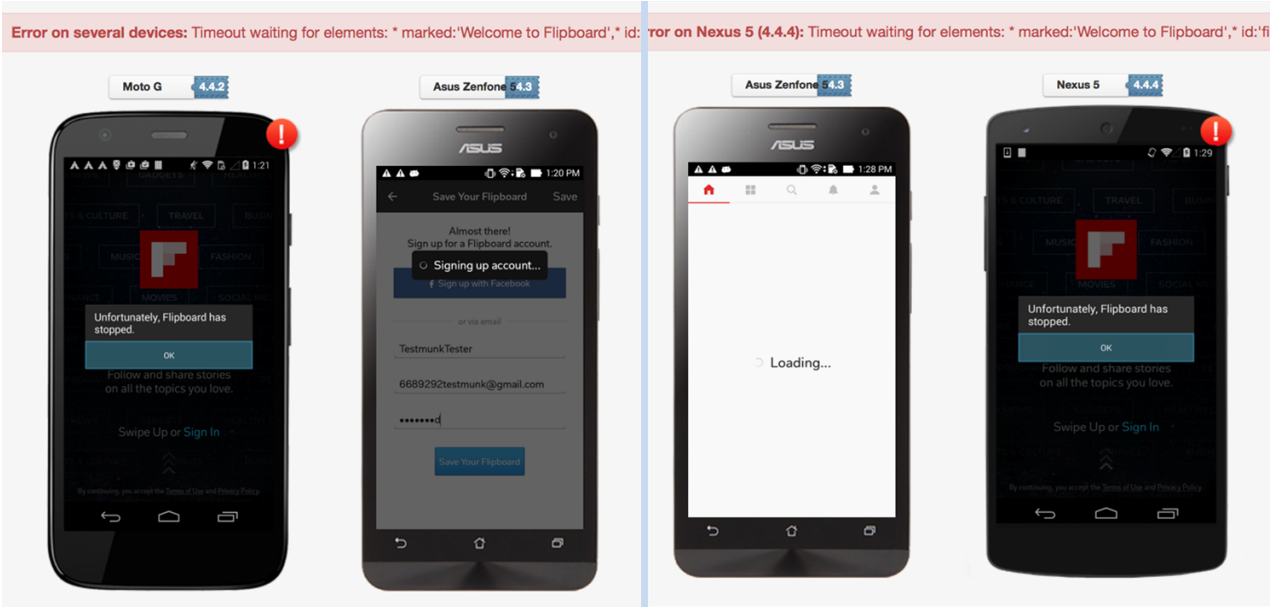
In a given testrun, not everything can be asserted. Take, for example, the below issue, where the signup button as presented was obscured by the keyboard, making it difficult

for a user to select the option. It is possible, I suppose, that an assertion could be made regarding the keyboard not obscuring the signup button. However, that would assume the problem was anticipated. An extraordinary number of assertions would be required if we wanted to verify that all buttons are placed exactly as desired on a particular screen/view for multiple devices. Rather than making the testcases per view overly long and complicated, we can instead focus on the top 3-5 assertions. With the related screenshots, we can visually detect potential issues even if the testcase as written cannot anticipate the problem.



Sign Up Button Obscured

Detecting Crashes



Left: App crash on Login **Right:** App Crash upon Clearing News Item from Main Thread.

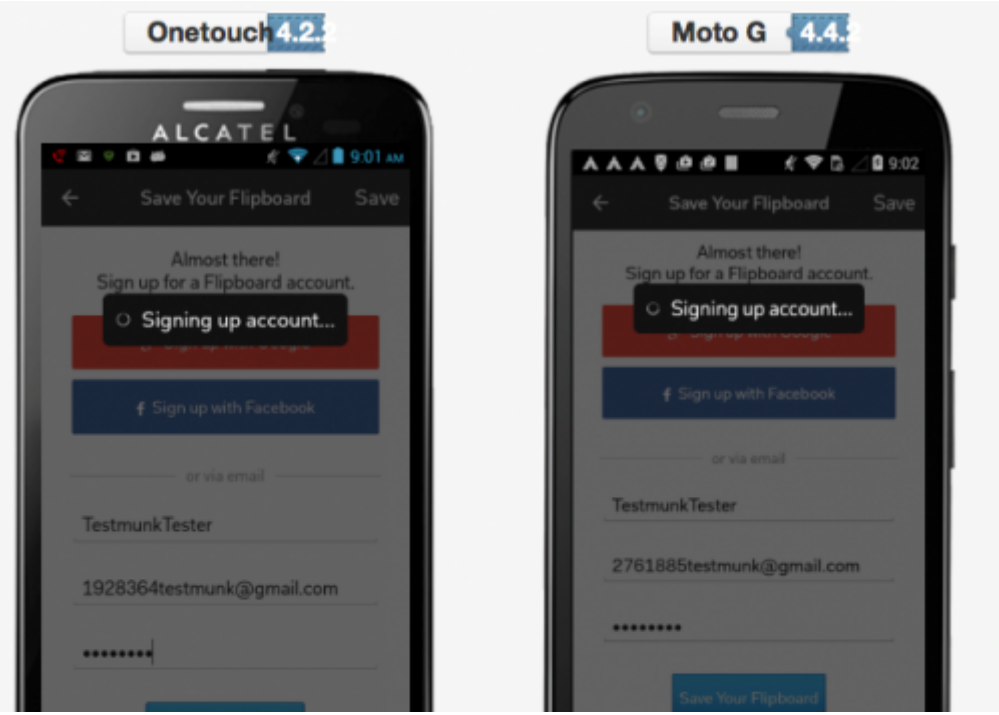
With our manual efforts, it was sometimes a hassle to provide the device logs that were routinely requested by the developers for troubleshooting purposes. Being able to quickly download the device logs from testmunk has made correlating test data and logs much easier. In the left example, we see the app crashing upon login. In the example above right, for instance, the problem was most likely that the main thread of the app wanted to clear a news item (or some news items) from the app cache. Since the cache could only be cleared from a background thread, the RuntimeException error was thrown, and caused the app to crash.

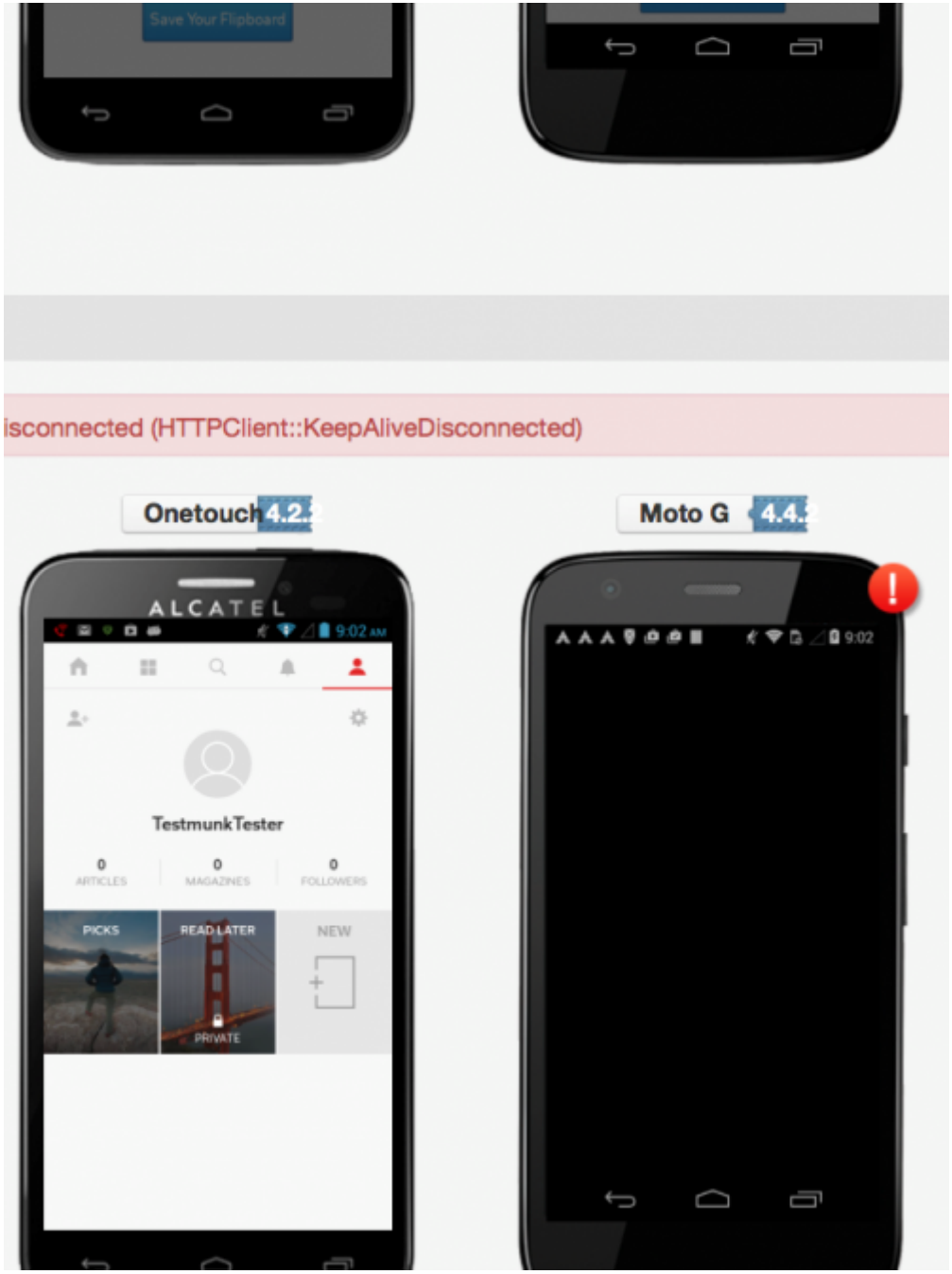
This particular error illustrates perfectly the point made earlier, as it was specific to devices running versions of Android greater than 4.4. This is why testing across different devices and SDKs is so important. Knowing what issue is specific to what version (or versions) is sometimes the best clue as to what might be happening. As a developer, it allows you to focus on differences in code between the affected and the functioning versions, and to apply fixes based on said differences.

Device Logcat files, captured by the testing solution, showed us the following info:

```
FATAL EXCEPTION: main
E/AndroidRuntime( 635): java.lang.RuntimeException: Only a background
E/AndroidRuntime( 635): currentThread = main
E/AndroidRuntime( 635): Main Looper Thread = main E/AndroidRuntime( 63
E/AndroidRuntime( 635): at flipboard.io.SectionDataCache.softRequireBa
E/AndroidRuntime( 635): at flipboard.io.SectionDataCache.clearItems(Se
E/AndroidRuntime( 635): at flipboard.service.User.removeSectionInterna
E/AndroidRuntime( 635): at flipboard.service.User.removeSection(User.j
E/AndroidRuntime( 635): at flipboard.service.User.unfollowSection(User
```

Another such issue discovered was an app crash issue specific to the Moto G. OneTouch, a very similar device running a slightly older OS version, passed with flying colors.

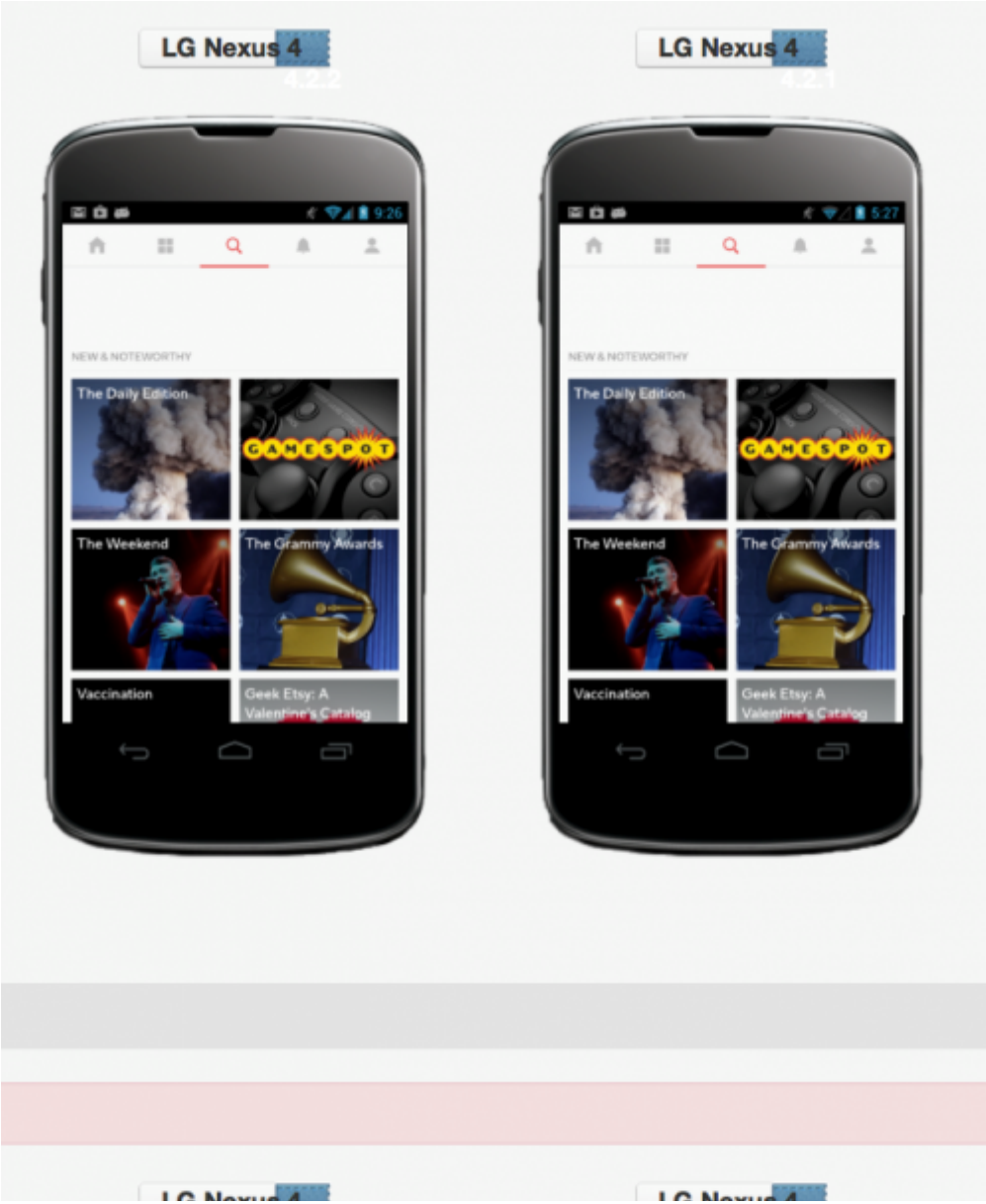




Moto G OneTouch Specific Issue

Here we see a critical issue occurring while entering text into the main search field. This

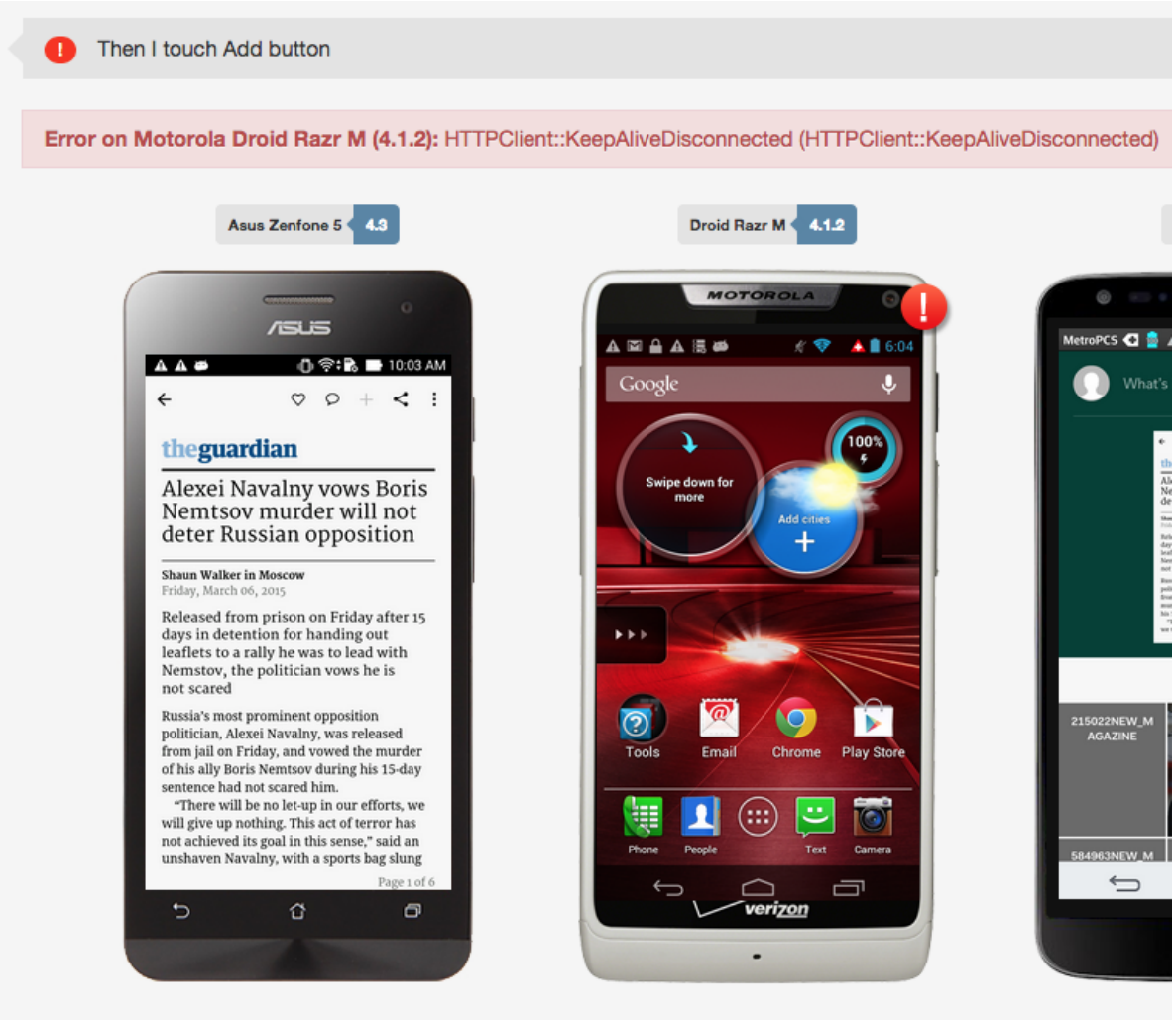
issue was likely the result of an incorrect Gradle (Android build system) setup. Sometimes, with this type of error, simply rebuilding the project can help. Otherwise, a check of the libraries used might reveal the problem. As always, quick discovery of the issue, and determining the scope of the issue (what devices are affected) is key to resolving it.



Possible Incorrect Gradle Setup

```
Log:
I/ActivityManager(25152): Force finishing activity ActivityRecord{434c
E/BufferQueue( 161): [Starting flipboard.app] drainQueueLocked: Buffer
I/dalvikvm(16104): Could not find method android.webkit.WebView.set
WebContentsDebuggingEnabled, referenced from method flipboard.app.Flip
```

Here, we discovered a very rare bug that may be related to incorrect use of Canvas class. Again, use of the associated LogCat logs proved very helpful in troubleshooting the issue.



Fatal Signal 11 Error

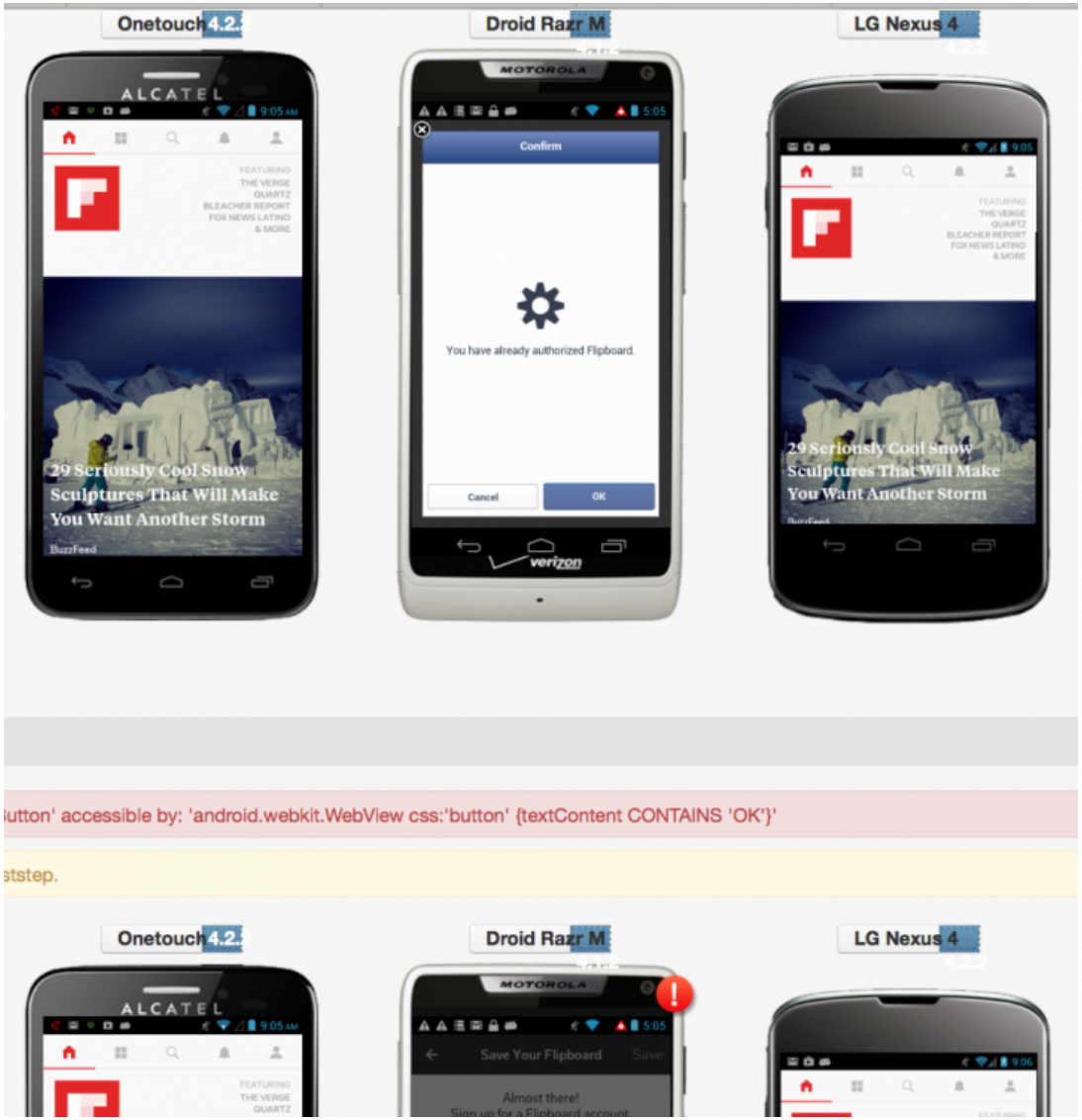
More info about this particular “Signal 11” error can be found on StackOverflow.

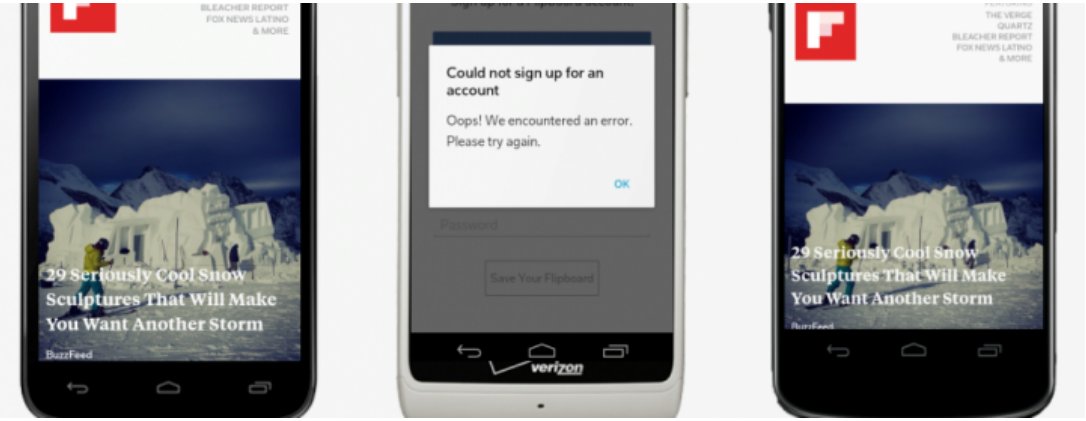
Other issues

Some issues do not fit neatly into any particular category, or the root cause is not

immediately apparent. In medical terminology, these might be termed idiopathic. Here, we'll stick with "Other issues".

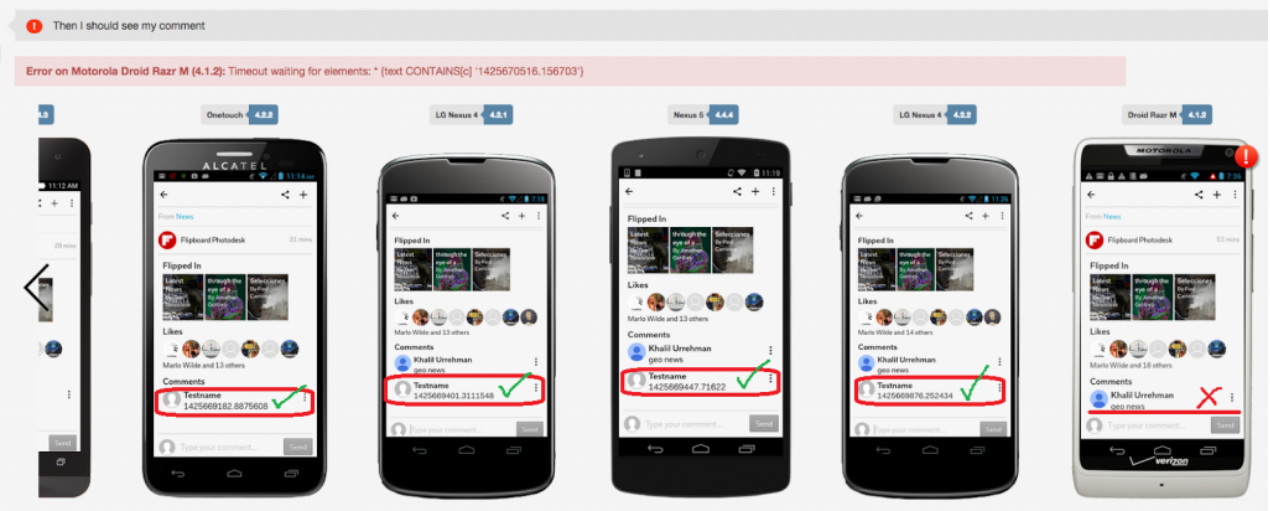
Below, we see a 'Sign Up' issue specific to the Droid Razr M device. For some reason with this device, the app appeared to assume a login and confirmation had occurred, preventing the teststep from providing the correct info. The app could not authorize based on the faulty (non-existent) information, and of course, threw an error.





Razr M Faulty Login and Confirmation

Another such device specific bug (most commonly occurring again on the Droid Razr M), was an issue where after adding a new comment, it was not visible on the screen.



Comment Missing

The testrun in which this issue was found was based on the below user story. The bolded text represents the failed step. As you can see the report generated by the automated testing solution is very helpful in determining exactly where the issue occurs. Here is the testcase that covered this user story:

Scenario: Comment on an article

```
Given I sign in with email: "testuser2@gmail.com" and password "tes

When I open one of the articles on the cover

And I add random comment

And I swipe Social screen up

Then I should see my comment

Then I remove my comment

Then I should not see my comment
```

Conclusion

Adding Automated Functional Testing to our existing unit and integration testing has proven one of the best QA decisions Flipboard has made, allowing the scope of what can be accomplished in a development cycle to grow. What used to take several hours of manual testing can be accomplished in minutes, allowing us to find more issues, focus manual testing on the most important of issues, and give developers more time for resolution of the most pressing issues. Adapting to the use of automated regression testing has given us an additional safety net, especially when it comes to the hectic times shortly before the release of a new app version. I'm sure as developers you have been in the situation where one or more fixes that were considered complete could not be included because there was not adequate time to test. Testmunk allows us to include (and test) last minute tweaks and fixes, or to resolve many of the last minute bugs

preventing release of a new feature. No longer do we have to deny a feature or shift a user story to the next release simply because we don't have time for another full regression test. Personal experience has me assured that Flipboard, having tasted the benefits of automated functional testing, could not do without it.

About the author:
Derrick Lam works as QA for Android at Flipboard, and has worked for over fifteen years in the industry. Living in San Jose, California, he has worked for companies such as Box, Color, Netflix, Apple, and Minerva over the years, helping to improve their applications. In his spare time, he loves to spend time with his family, and his daughter often twists his rubber arm to take her to the area's many theme parks.
Follow Derrick Lam on Linkedin

Read next

Download Testcases from the Wizard, and More Updates

Posted by Michael Walsh on April 29th, 2015

Weekend Readings

Posted by Michael Walsh on April 10th, 2015

Subscribe

to more geeky testing topics

Your Email

Contact

650-284-7139

hello@testmunk.com