

PERSPECTIVES ON AGILE SOFTWARE TESTING

*An anthology of essays
on testing approaches,
tools and culture by
testers for testers.*

Share this ebook.




















Contents


























A Timeline: The Evolution of Testing Tools	4
Is Selenium Finely Aged Wine? - <i>Anand Bagmar</i>	9
Testing in an Agile Environment - <i>Daniel Amorim</i>	15
Testing for Mobile - <i>Fabio Maia and Alabe Duarte</i>	20
BDD Style of Testing in Mobile Applications - <i>Prateek Baheti and Vishnu Karthik</i>	24
Continuous Delivery for Mobile Applications - <i>Gayathri Mohan</i>	28
Challenges in Mobile Testing - <i>Vikrant Chauhan and Sushant Choudhary</i>	37
Three Misconceptions about BDD - <i>Nicholas Pufal and Juraci Vieira</i>	43
Hiring Selenium QA People - <i>Paul Hammant</i>	48
Recap: Five Takeaways for the Modern Tester	54


















What's this e-book about?

























It's been 10 years since Selenium was born. The industry has changed tremendously since then. We want to recap the big developments of the last decade and introduce this anthology about testing methods and tools - some new pieces, and some from our greatest hits.

A TIMELINE: THE EVOLUTION OF TESTING TOOLS

Technology	 TOSCA TestSuite™ by TRICENTIS	OATS	 TestComplete	LoadRunner	QTP	Rational
Created			1999	2000	2001	2002
Open Source						
Commercial Support						
Web						
Mobile						
Selenium Ecosystem						

Technology	SilkTest	Watir		 <small>The Tester's Web Automation Tool</small>	WaitN		
Created	2005		2004	2005		2006	2008
Open Source							
Commercial Support							
Web							
Mobile							
Selenium Ecosystem							

Technology	CodedUI	Watij		Selenium 2 (Webdriver)	 BrowserStack	
Created	2010			2011		
Open Source						
Commercial Support						
Web						
Mobile						
Selenium Ecosystem						

Technology						
Created	2012		2013		2014	
Open Source						
Commercial Support						
Web						
Mobile						
Selenium Ecosystem						



Anand Bagmar

SOFTWARE QUALITY EVANGELIST — Anand Bagmar is a hands-on and result-oriented Software Quality Evangelist with 17+ years in the IT field of which 14+ years in the software test field. He is passionate about shipping a quality product, and specializes in building automated testing tools, infrastructure and frameworks. Anand writes testing related blogs and has built open-source tools related to Software Testing – WAAT (Web Analytics Automation Testing Framework), TaaS (for automating the integration testing in disparate systems) and TTA (Test Trend Analyzer).

IS SELENIUM FINELY AGED WINE?

How did it all start?

Agile methodology has been around since 2001. As teams started understanding more than just what's written in the Agile Manifesto, they started to identify the different practices that are required to be truly agile.

One of the practices that teams need to do well when working the agile way is Test Automation. Test Automation may seem easy - but in order for it to be truly effective, the team needs to have a lot of discipline in defining their process, choosing the right tools and technologies to give that quick feedback, and also allow the Test Automation to scale.

That said, even today, completing Test Automation in the same iteration along with development is a huge challenge for most. The inability of doing this becomes more apparent when Test Automation uses tools and technologies that are difficult to keep in sync with the rapidly changing product.

It was 2004 when Jason Huggins, while testing an internal application at ThoughtWorks, created a JavaScriptTestRunner that changed the way automating the browser (browser-based-testing) is done. This then evolved into "Selenium" which was then made open-source. Read the "Selenium History" for more details on how this came into being.

There have been, and in fact still are, many proprietary and commercial testing tools /products that have the USP of do-it-all 'automagically' (it = automation). Unfortunately, these tools / products come with a very high tool license cost.

This resulted in organizations / teams restricting the usage of these products to only a chosen few, in order to keep costs in control. This further created more silos in teams.

Selenium, being open-source, challenged this way of working and broke down the barriers.

All team members (QAs / Developers / ...), who knew programming, could now contribute to building and running Test Automation without having to worry about the license costs.

More importantly, the Developers and QAs could now talk the same language in the test implementation, thus reducing bad assumptions.

It's now been a decade since Selenium has been around, and a lot has happened in this time. In the technology space, more so in testing-based tools / technologies, that is a very long time for something to not just survive, but keep getting better and richer with age.

If Selenium was compared to a wine, then is 10 years long enough to consider it a finely aged wine?

continued ...

Current reputation of Selenium in the industry:

Today, it is extremely rare to find someone involved in browser-based testing who does not know about, or has not heard of Selenium.

- There are many other frameworks implemented in various languages that build on Selenium and offer more capabilities for Test Automation
- There are various innovative companies that have built products on top of Selenium to offer more value to its clients, such as SauceLabs, BrowserStack, etc.
- Service organizations across the world have the top layer of management talking and selling Selenium-based offerings to prospective clients
- There are numerous organizations that have sprouted over the years, selling Selenium-automation-based testing services
- Selenium is one of the top-searched skill-set in job profiles (related to Test Automation)
- The Selenium team is working hard and fast to come up with more capabilities, features, and also standardize browser and mobile-browser automation support

Why is Selenium so popular?

So how is it that Selenium survived for a decade and is getting stronger and more popular? Well, there are many reasons for this: the most important reason – it simply “works”. It does what the APIs claim to do.

It is open-source – so you can dig deep to understand how it works, and if it needs to change, the power is in the hands of each individual in the community. You don’t need to be in the ‘elite-class’ to have access to the core workings of the same.

The community support is awesome. If the Selenium core contributors cannot answer in time, there is vast amount of expertise in the community.

It is quite possible that someone has encountered something similar, and the ideas / suggestions / workarounds come in fast numbers. Also, fixes do come in very quickly.

It is free – so other than having good programming language expertise, there is no other ‘tool-cost’ involved.

Given the growth in adoption of Agile, Selenium-based automation fulfills all the requirements of teams needing to do Test Automation.

continued ...

Is Selenium the 'silver-bullet' of Test Automation?

As is the case with any tool / technology, you have to use Selenium in the right context. If that is not done, you not only start losing value, but also start facing many more challenges in the work.

As the agile methodology keeps getting more mainstream, the need for better and quicker Test Automation is apparent.

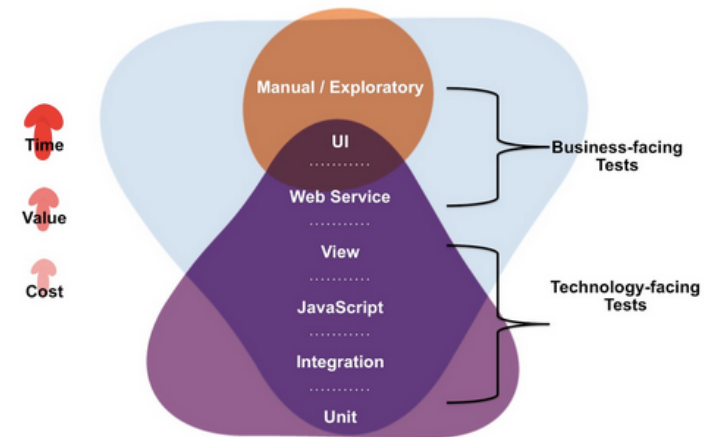
Amongst other reasons, Selenium IDE is a great way for people who don't know how Selenium works, to use the recorder and see what scripts are generated to interact with the browser. However, like with any recorder-based tool, you need to invest some time in making sure the recorded script is going to work in all situations, and that you will be able to use different types of data (if that is a requirement), handle waits and timeouts intelligently, etc.

Unfortunately, not many take that extra step. Some of those who knew how to record a script using Selenium IDE started claiming Selenium-expertise.

Selenium based automation seemed to be 'the right solution for ALL test automation problems. It became the silver-bullet for automation. However, there is no such thing as a 'silver-bullet'. We tend to forget that the Test Pyramid has multiple layers, each representing a different type of test which is appropriate for the product-under-test.

Here is what an ideal test pyramid might look like:

IDEAL TEST AUTOMATION PYRAMID



It is very important for the Selenium-based (or for that matter, any other UI-Automation-tool-based tests) to focus on top of the pyramid, and not try to do-it-all. That is where the true value of these tests is.

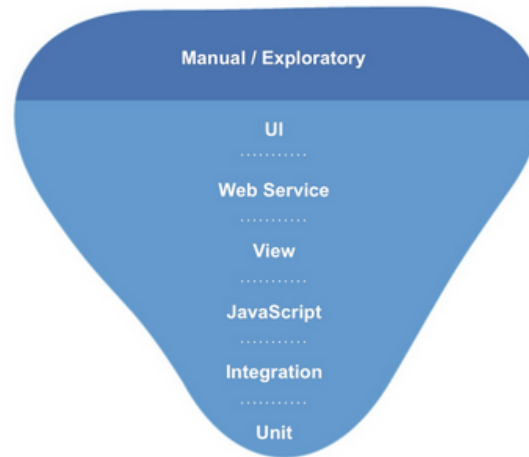
People who had no idea, or a limited idea about how a Test Automation framework should really be built, became Owners and Test Architects of Test Automation. Adding to the woes, delivery pressure, and the number-driven goals of Testing and Test Automation resulted in just adding automated tests quickly, without proper thought or practice.

continued ...

So very quickly, the ideal test-pyramid shown below changes to one of the anti-patterns:

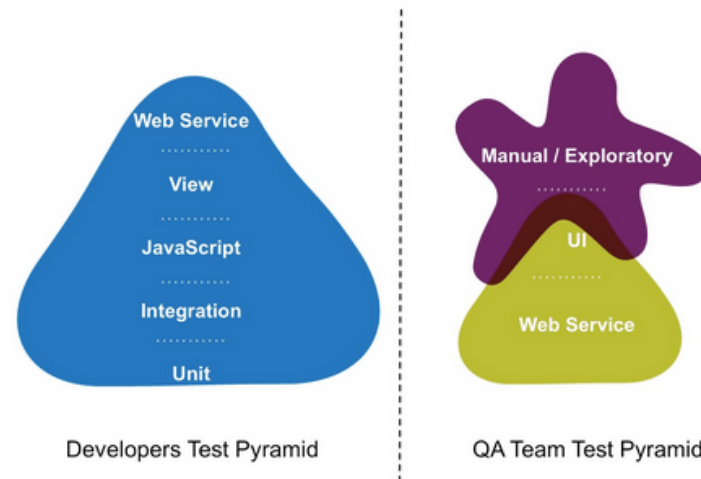
ICE-CREAM CONE ANTI PATTERN

Anti-pattern #1



DUAL TEST PYRAMID ANTI-PATTERN

Anti-pattern #2



continued ...

As a result, the silver bullet became the just a bullet, a lethal one - teams losing confidence in automation, its results, and its team members!


I have come across so many cases where teams have thousands of Selenium-based automation tests - and the maximum time they spend is in maintaining / fixing the tests, while there is an army of 'manual-testers' running regression testing against the product. This is undoubtedly a lose-lose scenario, but a very real and uncommon one. Then - the blame game and finger pointing starts.

It is very important to build an Automation Framework that can live and grow (= evolve) along the journey for the duration of the product-under-test. It has to be maintainable and scalable - which is an unsaid requirement for testing enterprise-level products. At the same time, it is very important to identify the 'right' tests for automation via the browser.

So, what is next? What does it mean for you?

The Future of Testing is very bright. We have a plethora of devices and technology advancements happening every minute. It can seem scary, (and for a good reason too), to those who are not on the path of learning and adopting what's next in science and technology..

Here are some thoughts of what you can do to get on that path:

- The role of a QA has become much more complex and requires a mindset shift as well as technical disciplines
- Get involved and understand / contribute towards product architecture - this will help understand what needs to be tested, what needs to be automated at which layer of the test pyramid, and why
- Understand that Test Automation is a form of Development. Get proficient at developer-practices, and apply your testing-hat to become great at Test Automation
- While websites had become the challenge to tackle 10 years ago, QAs must now understand how to automate testing against not only websites, but also mobile and interactive applications
- Behavior-Driven Testing (BDT) is one of the ways how you can identify and build a good regression suite to avoid getting into the anti-patterns mentioned above. 



Daniel Amorim

AGILE QUALITY ANALYST — Daniel works as an Agile Consultant QA at ThoughtWorks. He's been working in the IT market since 2007 and is pursuing agile testing best practices in order to contribute with the development team, not just for automating tests, but also to help build high quality applications through a collaborative agile team work.

TESTING IN AN AGILE ENVIRONMENT

Agile Testers are often known as Quality Analysts (QA), Software Engineers in Test, Test Engineers and QA Leads, among other variances. I've been working as an Agile QA for a while and I would like to share my point of view about how QAs work in an agile team. In this article, I will use the term QA to represent an "Agile Tester".

Most people, even in agile teams, treat QAs as a sub-role or a separate role in the team. I believe this is an outdated concept. The difference between a QA and a Developer lies in the mindset.

But what distinguishes QAs amongst themselves? QA profiles can be sorted into three categories: Business, Technical and DevOps. I call these the "three dimensions of a QA profile". QAs can have either one of these different profiles, or combine them according to the level of knowledge they have in each dimension.

Let's dive a little deeper to understand each one of these dimensions:

Business dimension:

The QAs in this dimension are really business driven. They have skills that will help their team understand the business context given by the client.

They have good communication skills that will facilitate the team to focus on the business problem during the entire project.

Extracting acceptance tests from clients is one of their specialities and BDD is one of the techniques they use to break down barriers between business context from the client side and technical context from the engineers' side.

They pair with developers to huddle with the client before they play stories, in order to get enough information for the story to be played. During this period, they drive the pair to write acceptance tests to make sure that the story is tested before they move it forward.

continued ...

These QAs usually read books such as:

Specification by Example: How Successful Teams Deliver the Right Software

Bridging the Communication Gap: Specification by Example and Agile Acceptance Testing

The Cucumber Book: Behaviour Driven-Development for Testers and Developers

Technical dimension:

I relate to this dimension closely because the QAs here are really technical and have good programming skills. Ideally, there shouldn't be any difference between a QA and a dev. In an agile team, everyone is an engineer and should be treated as such.

The technical QAs pair with devs to build the application without a technical gap. They code together as well. They also support devs to do TDD, fostering good practices for clean code and design patterns, and ensuring high quality code.

They have a lot of knowledge on test automation and help the team choose the best test frameworks for the project. They are also responsible for making sure the team has a good test strategy in place.

The QAs in the technical dimension may also work on performance tests and security tests, depending on how advanced their knowledge on non-functional tests is. For performance tests, they work with the client and find out SLAs (Service Level Agreement). They then create performance tests to measure and track the improvement done on the application related to these SLAs.

They also create tests to ensure that the possible vulnerabilities are being covered for a security mechanism.

These QAs are also involved in security. They understand the business context from the client and analyze possible vulnerabilities.

These QAs usually read books like:

Test Driven Development: By Example

Clean Code: A Handbook of Agile Software Craftsmanship

Selenium Testing Tools Cookbook

continued ...

The Art of Application Performance Testing: Help For Programmers and Quality Assurance

DevOps dimension:

How is DevOps related to testing? There are a lot of things that QAs can do to help teams through their knowledge of DevOps.

They introduce the practice of continuous delivery and help the team create a continuous integration pipeline in order to get faster feedback after each commit. This helps the team deploy new features to production more often. In some cases, each commit goes directly into production right after successfully passing through the pipeline.

This pipeline will also run the build and pack code quality tool, unit tests, component tests and functional tests.

The DevOps QAs set up scripts for the team to easily run tests in their local machines. In some cases, virtual machines are required and here the tests are set up to run in parallel.

They use task runners for the team to execute repeatable tasks easier, like auto watches to run tests automatically after each time the developers save the code. It decreases the time of the feedback during development.

This QA usually reads books like:

Continuous Integration: Improving Software Quality and Reducing Risk

Continuous Delivery: Reliable Software Releases through Build, Test and Deployment Automation

What's common for all these QAs?

QAs in all three dimensions keep the team focused on delivering the right value for the client during each development cycle. At the same time, they are concerned about the quality of the product that is being delivered.

continued ...

Along with sharing ownership of the tests with the rest of the team, they share their knowledge about these tests. Through this approach, every single team member thinks about tests regardless of their role. QAs wear many hats, but their main focus should be to help their team deliver business value frequently and with quality.

A book that all Agile QAs should read:

Agile Testing: A Practical guide for Testers and Agile Teams

How about you? Where are you on these dimensions? What do you want to get better at?

Originally published at: <http://www.thoughtworks.com/insights/blog/agile-tester-30>



Alabê Duarte

DEVELOPER

Fábio Maia

DEVELOPER

TESTING FOR MOBILE



Writing tests for applications these days is more than just a formality of sorts. We understand that testing directly influences the design, quality, maintenance and evolution of the software service or product. It also influences the team's confidence.

Testing has become an essential requirement of agile methodologies and practices, such as continuous delivery and various flavors of Driven Development (xDD). The tester's toolkit has exploded. We have mature tools that can extract behaviors from web frameworks and feed them into our code as unit tests.

What about mobile applications? How does writing tests for mobile in particular differ? The mobile approach has a tiny difference when it comes to writing tests. If our mobile application consumes some web resource, we'll probably fetch the content asynchronously for a better user experience.

What should happen if the user answers a call while using the app, and then returns to the app after that?

Should the app keep the user's preferences in-sync with the application's main web page during that time? What about internet latency, 3G/4G connections, WiFi and geolocation issues?

How can we apply mobile test API's or test suites, the strategies that we use to avoid waste and ensure continuous delivery in web apps? Should only the native language be used for write those tests?

Let's start addressing those queries!

Unit tests:

Unit tests validate each unit of functionality from our application and therefore comprise most of the total test sets. It is this set of tests that developers use, in order to get faster feedback and know if the application is working as desired. So when writing a unit test, we must keep in mind that it should run fast.

In order to test web applications and keep unit tests running fast, we often use Test Double. It helps us avoid the code to communicate with any process or external dependencies that are not in our control.

In mobile applications, it's very common to make remote APIs calls, access file systems or communicate with another applications from the device, e.g, the application that maintains the user's contacts. For this, we use a Test Double when the code is interacting with any of these components.

continued ...

Another good practice is to use Test Double when the code we are testing calls any API method from the mobile framework we are developing.

To facilitate this, it is advisable to separate our application logic into units that function independently from the mobile framework we are using.

For example, if we either write a subclass of the Activity class in Android or UIViewController from iOS, we should try putting the main logic of our application outside of these subclasses.

If we don't do this, we can face difficulties like having to trigger the initialization events that are part of the life cycle of these classes within our tests, to make everything working properly.

Let's assume that one of the screens of our application is listing the last 10 commits from GitHub, instead of our LastCommitsActivity to create a class that is responsible for:

Controlling what should be displayed in the ListView

Controlling the touch event of an item from the ListView

Making HTTP requests to retrieve the last 10 commits

While the first 2 items are completely dependent of the Android APIs, the last item is the real feature of this screen and it could be moved out to a new class that might be called LastCommitsService that is composed with LastCommitsActivity.

Thus, if our application logic is split up, we can worry less about setting up or stubbing out objects properly from the Android API, and devote more time to ensure that the piece of the functionality that should be tested works the way we want.

Moreover, the time it takes to initialize a new activity on our unit test can cause the tests not being as efficient as they should be. After all, aren't we seeking faster feedback?

You will understand better below how you can create tests for the first two items from the example.

Integration tests:

For testing integration points from the components of our application, the integration tests end up being slower than unit tests.

continued ...

In the context of mobile applications, they are the ideal tests for verifying that the application works when it interacts with native objects like the ones we use from the framework to which we are developing.

For example, in iOS we may want to check if a new UIViewController is created that's responsible for displaying detailed information about a song when the user chooses a new song from our list is being handled by a UITableViewController.

Although there are integration test tools in the language of our chosen framework like KIF for iOS or the native Android testing tools, there are other possibilities that can be more interesting.

A very interesting example is Appium that allows us to write tests for mobile applications in different languages in which our application is being developed.

This magic happens because Appium implements the Selenium Webdriver API, where any other language can connect to it through a webdriver client and our application can be running either on an emulator or directly on the device. ■



Prateek Baheti

DEVELOPER — Prateek Baheti is an application developer at ThoughtWorks who has worked on Twist for over 2 years. When not coding, he enjoys driving, listening to music, watching cricket and playing table tennis.



Vishnu Karthik

DEVELOPER — Vishnu Karthik is a developer at ThoughtWorks. He has been working on Twist and test automation. He has previously worked on health care services for Bihar (MoTech). When not coding he can be found playing ultimate frisbee.

BDD STYLE OF TESTING IN MOBILE APPLICATIONS

Mobile applications are pretty common these days. And most of the applications offer support for 3 popular platforms, iOS, Android and Windows phones, with Firefox OS as an emerging contender.

Functionality for these applications typically is the same across all the platforms. There will be differences between the platforms, like for instance, the way they each handle notifications. Testing such applications and ensuring that they work successfully on all the supported platforms can be a challenging task. Test cases have to be rewritten for each of the platforms and executed separately. This way, the test logic ends up being fragmented across different applications and each will have different levels of test coverage. The platform level differences are not related to the application's functionality, and the tests should ideally be expressed in pure domain terms.

The Behavior-Driven Development (BDD) style of testing can improve this situation dramatically.

Why BDD?

In the BDD style of testing, test specification is expressed in a neutral language which is very domain-specific. This approach helps provide a better insight into the test cases. Tests are expressed as high level logical statements and won't contain lower level details like clicking a button.

BDD assumes tooling support, which can link the test specifications to the underlying implementation. This style of testing has proven better maintainability and expressiveness of test cases.

With mobile apps in particular, BDD helps in the following ways:

1. Abstracts the implementation and provide high level steps:

BDD abstracts the lower level implementation details and provides higher level steps which will be platform neutral. Consider the following test scenario for a messaging application. The tests are written using Twist.

Messaging

Messaging:

- Login me as "foo@foo.com"
- I am sending the message "Lorem ipsum dolor sit amet" to "user@user.com"
- "user@user.com" should receive a notification upon new message arrival

The above test doesn't deal with any lower level details, instead it describes the test scenario in natural language with clear domain terms. In this test case, receiving a notification is a domain term and the implementation of it would be different across platforms.

continued ...

2. Thus test cases can be shared across platforms and teams:

A common interface will be used to communicate with different implementations.

```
public interface MessagingDriver {  
    void login(String userName);  
    void sendMessage(String toAddress, String  
message);  
    void checkForNotifications(String  
address);  
}
```

Now each platform can have specific implementations:

```
public class AndroidMessagingDriver  
implements MessagingDriver {  
    public void login(String userName) {  
        // login for android  
    }  
    public void sendMessage(String toAddress,  
String message) {  
        // send message for android  
    }  
    public void checkForNotifications(String  
address) {  
        // check for notification for android  
    }  
}
```

An environment variable can be used to switch the implementations at run-time and execute against different implementations.

```
public class MessagingDriverFactory {  
  
    public static MessagingDriver  
getMessagingDriver() {  
  
        String testPlatform =  
System.getenv("TEST_PLATFORM");  
  
        if (testPlatform.equals("android")) {  
  
            return new AndroidMessagingDriver();  
  
        } else if (testPlatform.equals("ios")) {  
  
            return new IOSMessagingDriver();  
  
        }  
  
        throw new RuntimeException("Failed to find an  
implementation");  
  
    }  
  
}
```

continued ...

Now that the test implementation will just have to use the factory and given appropriate environment variables set, the proper test driver will be picked up.

Test data and test scenarios need not be repeated, and they can be used as an executable documentation. Multiple teams working on the app for each platform can share these test cases. ■



Gayathri Mohan

QUALITY ANALYST — Gayathri is a Senior Quality Analyst at ThoughtWorks with 5+ years of experience. She has worked on multiple domains like Travel, Retail, e-Commerce and multiple platforms like .net, Rails, Android, iOS. She's worked on different verticals of testing like Performance, Automation, Web Services, Mobile Applications apart from manual testing. Recently her interests have pushed her into DevOps too. She's a frequent presenter at VodQA - TW India event for testers.

CONTINUOUS DELIVERY FOR MOBILE APPLICATIONS

Continuous Delivery (CD) is a set of practices, processes and tools to make a software application available for deployment/release at any point in time.

In this chapter, we will be looking at CD in the context of testing - manual and automated - Mobile apps. We will be touching upon CI and cross platform automation framework.

CD for mobile is more challenging than web:

Mobile apps have a lot of dynamic parameters both from development perspective, as well as testing. Though the size of the mobile app in terms of the number of features and screens may be less compared to web applications, it has a lot of variables that add complexity.

A single mobile app can involve many native device functions like camera, GPS, Wi-Fi, different hand gestures, screen orientations and so on. In addition to this, we have the complexity of multiple platforms and multiple versions in each of them.

Different device classes (phones and tablets) and hence different screen resolutions must be added into this mix as well. This increases the development and testing scope (unit tests, functional testing, regression etc.,) considerably, even for a simple feature.

When the Mobile app is a hybrid app, the testing scope is multifold compared to development, due to the ease in which it can be extended to multiple platforms.

The unit testing frameworks for Mobile apps are still evolving and are not always easy to work with. This may increase the risk of having an ice cream cone instead of the test pyramid, in turn increasing the regression testing cycle!

With all these challenges, CD for mobile is bit more challenging than it is for a web application which has been practiced and proven for a very long time.

CD is more essential for mobile:

Mobile apps see a pressing need for CD as the mobile platforms themselves see rapid development. New devices are released all the time. For instance, two versions of iOS - one minor and one major (6.1.6 & 7.1) - got released within a period of 2 months. Consider a project team in the above situation, which is all set for release to support versions till iOS 6.1 and all possible devices.

continued ...

During the development cycle, there is new platform release iOS 7.1! Now, the app needs to support one more version and cannot be released. The team may end up in an infinite loop of upgrading and testing their app on new OS versions and devices if CD is not practiced. This is a high risk, not only from the project team perspective but also for the customer's business.

Practices that help:

Though CD for mobile is a challenging task, interestingly, it is achievable through some good practices. Some of them are:

- Define devices, platforms and screen sizes supported clearly at the beginning of the project. Often, people forget that 'Android 2 & above' specification, translates to 7 different versions of Android
- Do market research on the most used OS versions and screen sizes. Surprisingly, that may not be the latest version. While JellyBean is most popularly used, Gingerbread has more users than Ice Cream Sandwich though the latter was released more recently
- Plan your releases as per above data. Plan short releases with two or three critical features on one platform at a time
- Use feature toggles for features spanning across multiple releases
- Create shorter requirements/user stories just enough to cover a maximum of couple of screens at a time. Though the screen is small and has less number of elements, there are more dynamics that need a keen observation
- Do devbox testing with physical devices. While simulators are going to be helpful for getting quick feedback, there are examples where the behavior is different on a physical device. For example, Wi-Fi on simulator is using the host workstation's internet connection which will simulate the best case scenario. But there is a possibility that we may miss the behavior of the app on a slow connection
- Choose physical devices in extreme configurations (big screen, small screen etc.), and also have a device with the most widely used configuration. In the web world, this is equivalent to testing on IE vs Chrome!
- Get feedback from a UX expert, if possible. This will add significant value as UX is unique for mobile apps
- Try to build an elaborate functional test automation suite as the current available automation tools can run really quickly on CI. Most of the tools provide APIs to automate device specific features like GPS, Wi-Fi etc.

continued ...

Nevertheless, look for possible alternatives that can be used to simulate the features that can't be automated. For example, barcode scanning can be automated by having a capability in the app when running in test mode to enter the actual digits and proceed. Though unit testing frameworks are still evolving, it is worth the effort to build an exhaustive set of tests.

Cross platform functional test framework for hybrid apps:

Hybrid Mobile Apps are mostly chosen for the ease and flexibility that they extend in different platforms. It is reasonably easy to create and maintain a cross platform test automation framework too! Let's start creating one such framework here, supporting both iOS and Android.

Let's say we have a sample hybrid mobile application which has a single page, asking the user to enter text in an input box and once entered, it just displays the text entered.

Mobile automation frameworks are no different from the web automation frameworks. We can use the typical Page Object pattern with a BDD layer on top, as it gives a common abstraction layer across platforms to automate this app. Choose an automation tool of your choice.

ThoughtWorks' Tech Radar gives a fair idea of tools to use, and also those to avoid. A fair assessment of the tool would be good to begin with, to check if it has appropriate APIs sufficing the application's testing needs.

Let's choose Calabash here as it provides support for both iOS and Android hybrid apps. We will be using Cucumber for the BDD Layer.

Step1: Defining the test intent:

First and foremost, let's define the intent of the test in a feature file as below:

```
/features/texting.feature
Feature: Texting
Scenario: Verify once I enter text, the text
appears on the page.
Given I see the texting page
When I enter "hello world"
Then I should see "hello world" above
```

I have used imperative style to describe the intent here than declarative to form a mental image of the app. Please check the benefits of using imperative vs declarative style here.

continued ...

Step 2: Implementation of the intent:

Once the intent is detailed, let's define its implementation in step definitions:

```
/features/step_definitions/
texting_steps.rb

When /^I enter "([^"]*)"$/ do |text|
  @text_page.start_texting(text)
end
```

Step 3: Create page object pattern:

Now, let's create our page objects and define the page members and actions -

```
/features/pages/text_page.rb

class TextPage
  include BasePageOperations

  TEXT_BOX=".textBox"
  def start_texting(text)
    enter_text(text, TEXT_BOX)
  end
end
```

We will be defining only a single step_definition and page object as above for both iOS and Android. This is because the elements in a hybrid app are normal HTML elements, except that they are packed in a WebView container in mobile. We can use the traditional CSS locators to define the elements.

Now, note that there is a method called 'enter_text()', which is neither defined in this class nor it is a Calabash method. So, where is it defined? We will be getting to it in the next section.

Step 4: Conditional loading:

Now we have our normal page object framework designed! But how do we make this work across platforms? Where do we call the platform specific methods?

Let's take a second look at the TextPage class. At the very first line of the class, we have a module called 'BasePageOperations' included. This module is where we call the platform specific APIs, which means we will need to have two different versions of this module, one for iOS and one for Android.

continued ...

Well, you may ask, “Won't Ruby get mad at me for confusing it with two modules with same name and same method signatures?” No, it won't, as we will be loading them conditionally. First, let's create two folders - *features/android-support* and *features/ios-support* and create the appropriate 'BasePageOperations' module within them.

```
#features/android-support/android_base_page.rb

module BasePageOperations
  include Calabash::Android::Operations
  def enter_text(text, element)

  performAction('enter_text_by_selector', "#{element}", text)
  end
end

#features/ios-support/ios_base_page.rb

module BasePageOperations
  include Calabash::Cucumber::Operations
  def enter_text(text, element)
    touch("webView_css: '#{element}'")
    keyboard_enter_text(text)
  end
end
```

Having defined the platform specific methods in appropriate places, we need to load them conditionally for each platform.

We can do that by simply requiring the appropriate folders in run time for different platforms in 'cucumber.yml' file as given:

```
#config/cucumber.yml

android: PLATFORM=android -r features/
android-support -r features/pages -r
features/step_definitions
ios: PLATFORM=ios -r features/ios-support
-r features/pages -r features/
step_definitions
```

Step 5: Running the tests across platforms:

Done. We've built our first test, now we want to see it running.

```
To run on Android:
>calabash-android run <apk-path> -p
android

To run on ios:
>cucumber -p ios
```

Scalability to new platforms:

Hybrid apps have two advantages over native apps from a test automation perspective. Hybrid apps have the same HTML elements on all platforms and most of the times, they have the same navigation steps across the app. This gives us the benefit of abstracting the platform specific code to the last layer instead of replicating the steps and pages for each platform.

continued ...

Now to extend this test suite to run on any new platform, say Firefox OS, all we need to do is define 'BasePageOperations' module for the new platform and our test suite is ready to run on the new platform! This also helps while upgrading tools, as all we need to change is in one place.

Continuous Integration and Mobile

CI is one of the most important and basic part of Continuous Delivery. Let us see how we can integrate our functional tests seamlessly as part of our CI.

CI setup for android:

For running our Android functional tests, we need the '.apk' file of the application to be deployed on the emulator, for every build. This means that the previous stage in our pipeline, let's call it 'android-build', needs to be configured in such a way that it passes on the '.apk' file as artifact after building the application code and running the unit tests.

Once we get hold of the '.apk' file, Calabash takes care of deploying it on to the emulator. Running the tests and viewing the results of the tests are going to be as simple as running few commands on the command line.

A point to be noted is that Calabash doesn't launch the emulator automatically before deploying. It looks for an active emulator of the appropriate version. This means that we need to have steps that launch and kill the emulator before and after the functional tests are run.

Headless emulators:

Once we create an emulator manually with the intended configuration, we can launch it headless with the following command:

```
>emulator -avd ICS -no-window
```

We can kill the emulators once the tests are done running, to save computer cycles.

```
>adb emu kill
```

Deploying and running tests:

Now to deploy the application, Calabash gives an API 'reinstall-apps' which can be placed in a Before hook in the code to ensure new deployment before every run. We can now run the functional tests by

```
>calabash-android run <apk-path>
```

continued ...

Reports and screenshots:

We can get a consolidated report of the functional tests run with error stack trace and screenshots attached by slightly tweaking the above command like below

```
SCREENSHOT_PATH=/features/screenshots/  
calabash-android run <apk> --format pretty --  
format html -o features/android-report.html
```

Overall the task setup may look like

```
export ANDROID_HOME='android_sdk_path'  
export APK_PATH= 'apk_file_path'  
emulator -avd ICS -no-window  
SCREENSHOT_PATH=/features/screenshots/  
calabash-android run <apk> --format pretty  
--format html -o features/android-  
report.html  
adb emu kill
```

We may have to re-sign the app once manually, before the first functional tests run. See Calabash documentation for more details.

CI setup for iOS:

iOS CI setup is slightly different from android. Calabash-iOS launches the simulator automatically and closes it once the test run is done.

We may have to set the APP_BUNDLE_PATH variable to the path, where the '.ipa' file of the application gets generated after the dev build. Once the path is set, Calabash launches the simulator, deploys the app, runs the tests and closes the simulator on its own. So our iOS task may look like:

```
export APP_BUNDLE_PATH='path_to_ipa_file'  
SCREENSHOT_PATH=/features/screenshots/  
SDK_VERSION=5.1 cucumber --format pretty  
--format html -o features/ios-report.html
```

There is a tricky part while using Calabash to run functional tests. It needs the 'calabash.framework' to be linked to the application code, which means that:

1. We need to run the functional tests in the same box where we have the developer code
2. We need to link calabash.framework to the code after every developer build as it could have done a 'clean'. If the clean is happening on every build, we may have to run 'calabash-ios setup' in directory where .xcodeproj file is present.

The Calabash tests, both for Android and iOS, do run quickly, which is an advantage over web automation. Hence we may choose to have the entire set of tests for each of the platforms as part of CI.

continued ...

Regression tests for multiple versions:

We can run the tests against multiple OS version, and even devices as part of a Regression test pipeline. The regression pipelines can be configured to run the tests on different versions of platforms as above just by launching the appropriate version of emulators and connected devices. However, we should note that there could be certain tests which need alterations for different versions. We need to tag those tests appropriately and run them in the right pipeline.

Multiple release management:

Now that we are ready for first release, prepping the next set of features for release is the task at hand. Feature toggling is a nice way to switch off/on a feature for current release while the development is still in progress.

We achieve similar results by using the tags feature of Cucumber. Mark the features which are going live as 'release_1', 'release_2' and so on and run them as needed. We can setup pipelines for different releases and run tests as below.

```
>calabash-android run <apk> --tags @release_1  
--tags ~@release_2
```





Vikrant Chauhan

QUALITY ANALYST — Vikrant is a technology enthusiast. He loves exploring new tools that would help him ease his day to day work. He's a keen follower of open source testing tools and tries to use them as and when possible.



Sushant Choudhary

QUALITY ANALYST — Sushant is a passionate technology lover and a sports enthusiast. Considering the ever changing technology scenario and its impact on the society, he is always keen on exploring and applying new developments in software engineering.

CHALLENGES IN MOBILE TESTING?

The number of people owning a smartphone is increasing, giving rise to a higher number of people using the Internet on their mobiles. Therefore, most organizations have begun to actively increase their presence on mobile, leading to a thriving mobile application development community.

Before we jump into the challenges, let's discuss briefly the various types of applications that you would encounter on a mobile:

1. Native App: Application that uses the native components for UI elements
2. Hybrid App: Application that uses a mix of native and/or HTML components for UI elements
3. Website : A mobile-friendly website.

While working on a mobile app for a client, we encountered a few challenges that we'd like to list out:

Mobile Website Testing Challenges:

1. Responsive web design :

With the rising number of OS, browsers, devices and screen sizes, we need to ensure that our mobile website design and usability does not break from a phone with 240*320 screen resolution to an iPad with 1024*768 landscape screen size. Mobile development has moved from a pixel-perfect web design principle, where the site is designed to resemble the mock-up as much as possible. Now, the objective is that mobile websites are equally responsive on all the devices.

2. Testing across a multitude of browsers (Chrome,Safari,Firefox) :

Before we start testing, it's important that we identify our combination of browser and devices to test mobile web. A little amount of data crunching sourced from the mobile traffic data and device stats helps a lot.

continued ...

Even with many service providers offering synchronized testing on multiple browsers, it remains a challenge to execute, analyze and report the testing for mobile website.

3. Simulator/Emulator vs real devices:

We've spent hours debating the use cases for testing on a simulator versus a real device. A simulator works very well for speed and responsiveness, but interactivity is best tested using a real device. The device can be used to test gestures and real world scenarios, like website tear down when interrupted by a call or a performance bottleneck.

4. Single page or pageless design:

It's the design of the future, as it is seamless, intuitive and helps in iterative development based on user feedback and analytics. With a plethora of libraries available to manage browser states and query strings, it poses a major challenge to the tester by exposing the website to page refresh, history of navigation and efficient content load/render problems.

5. Testing number of requests and data transfer (performance):

Since most mobile websites love native experience, they have heavy client side application model and this affects the performance.

Developers may apply best performance guidelines for mobile website like JS minification, gzipping, lazy loading or application cache for local storage. Testing and analyzing the impact of these changes may be tricky at times.

6. Automation challenges:

Automation of mobile websites may throw many challenges related to tool/framework feasibility, headless vs browser call, javascript engines across browsers/third party libraries, and other test automation design patterns.

Mobile Application Testing Challenges:

Everyone wants a slice of the app-pie! We need to ensure our app's design and usability ranking is at the top. Although apps go through stringent market evaluation criteria done by Google and Apple, the quality of the product remains in a tester's court. Let's look at some of the major challenges faced while testing mobile apps on different OS and devices.

continued ...

1. Customers do not compromise on design:

Every app we build is competing with many other high quality products, it becomes imperative that our design is at the top of the game. It needs to be intuitive and sleek. With development based on wireframes and mock-ups, it's always a challenge for a tester to wear the ui/ux designer shoes, sometimes do a comparative analysis and become the voice of a real customer.

2. Development framework impact on testing:

Development framework selection for an app is critical, not just for the developers, but for the testers as well. With numerous libraries and frameworks available for building a cross platform app with native interface, the approach to test the application also requires a switch. It could be a pure native or a hybrid development framework like calatrava, which implements shared client logic with native UI. With shared client logic, you can be sure of the behavior on all the platforms - mobile, web or a hybrid app.

3. Native vs hybrid approach:

An app could be a pure native implementation or might have some web views. The difference in usability and responsiveness can be observed, based on these components.

4. Device vs simulator dilemma:

As discussed under challenges for Mobile Web, simulator/emulators are equally applicable to apps as well. For many manual test scenarios, especially related to performance, responsiveness and sensors like GPS or accelerometer work best on actual devices.

Most of the other manual and automation testing can be done on simulator/emulators which are available for all major platforms like iOS, android, windows and blackberry. An app needs to be test driven both on device and simulator, to understand their usage.

UI Automation of these components also require different approaches, right from the selection of tool to script implementation phase.

continued ...

Adding another thread to this discussion is the importance of mobile cloud services which offer hundreds of real devices/ OS range on cloud to test drive any cross platform mobile application. Of course, we save a lot of manual effort with them but a word of caution here: they can burn a hole in your pocket. So many options and so much less time to give feedback!

5. Performance matters :

All that matters to end-users is mobile experience. Performance metrics are very important with low computing (processing and memory) resources on mobile. It may encompass UI based performance, back-end load testing and monitoring of key performance metrics. Simple manual testing may include testing on simulated n/w condition, under constrained memory/processing conditions and app launch or load time etc.

Tools for fine grained performance evaluation can also be employed. Fine-tuning the app for the best performance can be challenging.

6.Information is power, how to get it?

An app has the opportunity to be in the palms of a user for most of the day, and we don't want to waste that. The real challenge is how we collect the data on its usage, active user base, frequently used user flow, revenue/user etc. Analytics services like Omniture and Tealeaf are our best friends. They also help us drive our testing in the right direction with information on critical user paths and error prone areas. Extracting the right data and taking useful inputs for testing can be learnt over time.

7. Testing third party integration and location services :

Testing frequently used integration like Facebook and Twitter requires special treatment. Although the actual services are provided by their APIs, the client-side code and end-to-end functional behavior needs to be verified. What makes it difficult is outage and late or early code pushes - we need to know this right away. Mock out dependencies help here, and integration contract tests can be built to ensure correct behavior.

continued ...

A similar challenge is posed by features based on location services. Simulation is required for features like push notification, which uses geofencing or iBeacon which uses region monitoring.

However, at the end of the day, all these services should ideally be tested in real environments.

8. Test automation challenges:

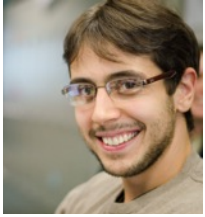
Developing the right automation infrastructure for mobile apps is critical, especially in agile teams. They need fast feedback, and this also signals their release readiness. Similar to web development, mobile also takes into consideration, factors like tools/libraries, test environment, test data management, mocking dependencies - the ultimate goal of building a right test pyramid.

There are lots of open source tools available for platform agnostic automation implementation. DSL support, hooks for native and custom controls, support for devices and simulator, language bindings, support for webviews, gesture simulation and test cloud can be a few important evaluation criteria to make the right choice.

9. Continuous integration and build distribution:

Automated build compilation and testing of apps with every check-in is important to deal with integration problems and it results in a stable cohesive codebase. With help of SCM tools like git/subversion, integrated with CI servers like Jenkins and GO, we can always have a production ready build, as it takes care of project compilation, setup environment, automation execution and archives the release artifact.

Before we pick a dev, qa or rc build for testing, we need to make sure that the build pipeline is not broken and code signing and obfuscation is correctly applied on the build. Making sure that production apps are signed with right distribution certificate is also important, as don't want to risk a rejection from Appstore. ■



Nicholas Pufal

SOFTWARE DEVELOPER — Nicholas has had a strong interest in programming since his adolescence, when he started dabbling in writing open source programs. He is passionate about design patterns, good development practices and ways to improve an agile team communication/delivered quality.



Juraci Vieira

QUALITY ANALYST — Juraci is a software enthusiast, who started his career as a Tester. He was obsessed by automation, only to learn that quality is built in the software, by when he became a very passionate developer.

THREE MISCONCEPTIONS ABOUT BDD

BDD has been often misunderstood among developers, QAs and even BAs. We often hear of teams saying that their project is using BDD, but when we check it out, it turns out to be using only a BDD tool for test automation - and not the BDD concepts itself. So in the end, we hear people arguing about the tools, and not about the ideas that inspired the creation of those tools.

The output of that is a bunch of complaints that we see in blogs all over the Internet - people that start to reject the whole idea behind BDD, only because they have tried to use a tool without first changing their attitude towards software development.

We commonly hear of these top 3 complaints about BDD:

#1 The client doesn't care about testing

This is the most common complaint. It makes complete sense to state that, as what really matters for the client is software that suits the client's purpose. Starting a discussion on testing somehow seems to give business people the green light to tune out. Also the word testing unfortunately bears a negative connotation in the software development community.

But wait a second, we are talking about behavior driven development, and it has nothing to do with testing. Testing is something you can't do until the software exists. Test is verification, and here we are talking about specifications.

BDD is a design activity where you build pieces of functionality incrementally guided by the expected behavior. In BDD we step away from a test-centric perspective and go into a specification-centric perspective, meaning that this complaint was born misplaced.

#2 The client doesn't want to write the specifications

This is the second most used complaint. Let's break it down.

"The client must write the specifications by his or herself."

Whoever complains about this is assuming that the client is expected to propose a solution to his or her own problem - the very problem that your software was supposed to address. If the client writes the specifications she won't benefit from something called **cognitive diversity**. Cognitive diversity comes from groups of heterogeneous people working together.

continued ...

For instance, she needs the advice of engineers that know the technical aspects of the problem that he wants to solve. She also needs a QA to spot edge cases that no one else noticed or else she may come up with a solution that is much more complex than it needs to be.

It's unfair to complain about something that we, as the development team, are supposed to help our clients with.

"The client needs to use the tool to write specifications."

Not really. What the client really needs to do is to provide to the team information about the problem that he wants to solve and together they come up with concrete examples to lead the development process.

#3 You can achieve the same without a business readable DSL

This argument is commonly found among developers. Most of these developers state that there is no real benefit in having this new "layer" - describing behavior in business readable language - as it only adds complexity and make the test suite slow.

If we take a look at this complaint when working on a Ruby tech stack, this usually implies that instead of using Cucumber, you can simply use Capybara + RSpec to achieve the same results with the benefit of having a faster test suite.

```
feature "Signing in" do
  background do
    User.make(:email => 'user@example.com', :password => 'caplin')
  end

  scenario "Signing in with correct credentials" do
    visit '/sessions/new'
    within("#session") do
      fill_in 'Login', :with => 'user@example.com'
      fill_in 'Password', :with => 'caplin'
    end
    click_link 'Sign in'
    expect(page).to have_content 'Success'
  end

  given(:other_user) { User.make(:email => 'other@example.com', :password => 'rous') }

  scenario "Signing in as another user" do
    visit '/sessions/new'
    within("#session") do
      fill_in 'Login', :with => other_user.email
      fill_in 'Password', :with => other_user.password
    end
    click_link 'Sign in'
    expect(page).to have_content 'Invalid email or password'
  end
end
```

That's like comparing apples with oranges: they are both completely different things.

continued ...

The benefit of having a business readable DSL - such as feature files written using Cucumber in this case - has benefits beyond the nuts and bolts of development. It's not just about code; it's about improving the all-important communication within the team.

For instance, it's about having a BA collaborating with a developer and a QA, with all of them making improvements on that single file (the feature file) - which is a non-abstract way of presenting ideas for the software. Plus, they will be using their complementary cognitive capabilities to brainstorm together about the best path to transform a specification into fulfilled business needs.

A successful case using BDD:

How complicated would it be for you to explain to a three-year-old child how a bank transaction works? The same challenge applies during software development, as the client domain can be sometimes pretty cloudy and vague to the delivery team.

We need examples to understand. Realistic examples are a great way to communicate, and we use them often without even thinking about it.

Working with real world examples helps us to communicate better because people will be able to relate to them more easily.

We can easily realize that when we are working on a complex business domain. A good example on that is a previous project we worked on. The company was an investment bank, and, as expected in a domain like this, the terminologies were really complicated, making it hard for the developers to communicate with the BAs from the bank.

So, in an effort to communicate better, part of our process was to have a quick call with the BA before playing a card. This BA would then share with the developers, the feature file that he came up with, describing each of the concrete examples that he thought about.

As we didn't have any QAs in this team, it's important to mention that one of the developers in this session had to keep a QA mindset - trying to spot edge cases, improvements to the scenarios, etc.

continued ...

The other developer on the team would properly focus on the technical challenges to implement it, such as making suggestions to move a scenario to a code level specification, which offers a faster feedback. Everyone could also suggest changes to a scenario's step definitions, in order to make it more meaningful to everyone.

The benefit of having it is that the BAs increased their understanding on the technical challenges of the scenarios and the developers had a clearer idea of the business needs and therefore what really needed to be developed. Plus, whenever changes had to be done, having that single piece of information would make everyone on the team in sync about it.

Down the rabbit hole:

To sum up, the main idea behind BDD is that it's driven to prevent communication gaps. Everyone in the team is communicating better and more often, based on real world examples and not on abstract and imperative requirements.

We hope that this article helps to make people more aware of the benefits of BDD - and make it clear that tools are only something complementary to the full stack agile methodology.

If you want to go deeper on the subject, we strongly suggest the following sources:

"Specification by example" by Gojko Adzic

"The RSpec Book" by David Chelimsky

Dave Astels and Steven Baker on RSpec and Behavior-Driven Development

Gojko on BDD: Busting the Myths 

Originally published at: <http://www.thoughtworks.com/insights/blog/3-misconceptions-about-bdd>



Paul Hammant

PRINCIPAL CONSULTANT — Paul Hammant is in his mid-40s. He's worked for ThoughtWorks as a 'Principal Consultant' since 2002, and has been consulting since 1989. As well as being nearly always billable to clients, he generally helps ThoughtWorks' Open Source agenda. No, he does not get paid to work on open source! He's a UK citizen but lives in New York courtesy of a Green Card.

HIRING SELENIUM QA PEOPLE

Mixed skills

Ideally, people working with Selenium to automate the testing of applications will have some development skills in addition to their QA skills. It could be that they are an experienced developer and they are helping start a QA automation side to their employer's project. It could be that they are a QA who has used a number of tools (like QTP) and are now trying to apply those skills to a more accessible Selenium solution. Lastly, it could be that they are a manual QA in the past and this is their first foray into QA Automation.

We're going to try to speak to all of those types, and we will phrase this as if it were candidate selection for interviews to join a project team.

Levels of experience

Inexperienced:

The candidate in question has not used Selenium in the workplace, and may only have played with Selenium-IDE in advance of an interview (at the recruitment agent's suggestion perhaps).

They may have a resume that speaks of Quick Test Professional (QTP), Test Director, or even WinRunner from some years ago. They may alternatively have a resume that details manual testing of web applications, using or not using tools to the cataloging of manual test suites and execution of the same.

When meeting these candidates, you may find programming languages, technologies, libraries, frameworks listed, but be aware that the candidate may be talking of team experience rather than personal use. In order to identify them as inexperienced, you'll have to construct questions that can differentiate between team and personal experience.

Entry-level:

The candidate has used Selenium-IDE for recording scripts and playback. These suites may be saved to SCM, but could have been just on a their 'C drive' and implicitly not shared between team-mates. The suites could have been highly evolved, relative exhaustive and relied on by the project to validate applications prior to deployment.

continued ...

The candidate may not qualify for entry-level Selenium skills if they have just used Selenium-IDE sporadically to aid form filling as they manually test something. If you think they qualify for entry-level Selenium, they should be able to use it in front of you to test a site of your choosing.

On the advanced side, they may be running their test suites automatically from Continuous Integration tools, and have a high level of confidence and ability to move forward with confidence.

Perhaps they are using Selenium-RC to execute whole sets of 'html' Selenium tests.

(There is at least one friend of the Selenium project that has really pushed the 'html' Selenium tests space, and we're not wanting to cast his dev team as beginners with Selenium here)

The nature of HTML, JavaScript and CSS should be understood to a rudimentary degree.

Data driving tests experience may or may not figure at this level.

The candidate has some knowledge of the DOM, and for Selenium specifically, how XPath, CSS or by-ID selectors are used to test the application.

They are also going to be able to checkout, and commit back to one of a number of SCM tools. Most likely with a UI rather than from the command line.

When meeting these candidates, they should be able to use Selenium-IDE in front of you to test a site you choose, and work through tricky issues exposed when playback doesn't exactly do what the record function identified. Not just that, but they should be able to simplify the XPath expressions the recorder has made. Choose a website other than your corporate one for them to test.

Intermediate:

On top of Entry-level ...

The candidate has transitioned to WebDriver in the workplace. They will be able to speak to the differences between it and Selenium-RC.

continued ...

They are able to program in one of Java (including Groovy & Scala), Ruby, Python, Perl, C# (or VB.Net), PHP, or a supported 3GL language that matches your needs. They are also adept with unit-testing (JUnit, NUnit, TestNG) frameworks for their language, as well as appropriate use of assertion technologies.

They may have OO knowledge, or may be beginning to understand it. Without it, they are likely to be using their language in a procedural way (which can also be fine, if its relatively clean from a purist's point of view).

If they are procedural (not yet OO savvy) they must be able to demonstrate looping and appropriate conditional aspects of the languages they are experienced with. If they are in the territory of OO in terms of experience, they should be able to give examples of where the facets of OO apply to Selenium test scripts.

The candidate will be able to discern between alternate selector strategies to use the most suitable. They will also use Firebug (or alternatives) to help them, and perhaps also be adept with XPathChecker or XPather, though hopefully using them far less than Firebug.

That pages are loaded in increasingly strange orders (we're looking at you Web 2.0) should not be a hurdle to the intermediate Selenium developer. They will be able to demonstrate techniques to wait for page events such that the test scripts they make do not fail when the application is slower than normal.

This level of candidate should be able to respond to broken builds (we're all using CI right?) find the issues, and liaise with the application developers to fix things.

They can, most likely, arbitrate over merge conflicts with the SCM tool they are experienced with. Hopefully they are as savvy with the command line as they are with the SCM tool's GUI.

In terms of testing style, this candidate will be able to separate happy-path tests from regression suites and project related QA.

The candidate, in the interview, should be able to extend an existing WebDriver codebase one or two tests more.

continued ...

If the candidate has not used WebDriver, they may still qualify for 'intermediate' if they have developed large suites of test scripts with Selenium-IDE and gone as far as using UI-Element and other advanced features of Selenium-IDE.

When meeting these candidates, they should be able to use WebDriver to script a test that they may have started in Selenium-IDE. Be lenient on the actual nature of the build-script that kicks off their chosen high-level test class.

Advanced:

On top of Intermediate ...

This candidate has more programming experience. Specifically: Closures (Groovy, Python, Ruby) should be familiar to them and they know how to make or use them in a test script, and why to use them. Groovy's community knows these as 'Builders'.

Functional programming constructs are likely to be known by the candidate, though they may not be in a position to use them.

The tester-developer should be able to craft large suites from scratch and define architectures for WebDriver tests. The style of the codebase should be compositional (this over and above OO). They should be able to train any newbie to the team (from formal development, or from the QA universe) in the tools and techniques needed to grow the test-base, and keep it running.

The storage of test-run results in (and otherwise use of) databases is a skill this candidate will have, as is the integration of related technologies for things that are suitable for them (like Selenium-SoapUI).

Testing architectures that adheres to behavior-driven testing (Cucumber, JBehave, NBehave, Cuke4Duke, RSpec and more) may figure at this ability level. The candidate will know when to use them, and when to stick with Unit-testing type technologies.

The configuration of a cloud capability for their Selenium scripts to use, should be within their skills. These could be Selenium-Grid, Sauce Labs, or BrowserMob (though these are all for different purposes).

continued ...

The candidate, in the interview, should be able to start a WebDriver project, write a build-script, get it all under source-control, hook it up to a pre-existing CI machine and keeps things appropriately separated within the project. Indeed, this level of QA should also be able to pass objective tests for regular developers.

Expert:

On top of Advanced...

The candidate will be able to build reusable frameworks and libraries that use WebDriver that introduce real value (rather than alleged value) for their team.

They will be able to debug into Selenium itself and provide contributions back to the Selenium team. They are more than likely to be a master of more than three 3GLs.

The candidate's name might be Simon Stewart, the author of WebDriver and an ex-ThoughtWorker. ■

Originally published at: <http://paulhammant.com/2011/08/04/hiring-Selenium-quality-assurance-people/>

RECAP: FIVE TAKEAWAYS FOR THE MODERN TESTER



The role of a QA is now more complex than before - get used to wearing many hats at once



Get involved and understand product architecture



Get proficient at developer-practices because Test Automation is a form of development



Learn how to automate testing against not only websites, but also mobile and interactive applications



Identify and build a good regression suite using Behaviour-Driven Testing, to avoid getting into the anti-patterns

STAY IN TOUCH

Sign up for our Perspectives Newsletter.

We'll periodically send you a newsletter filled with great educational resources, thought leadership and event information.

Sign Me Up

Share this ebook.

