Shashikant Jagtap - Hands On XCUITest for iOS apps

Shashikant : Hello everyone. Welcome to Automation Guild Conference session on Getting Started With XCUITest. My name is Shashi. And in this session we'll be going through historical testing in iOS World. Then we will discuss rise of third party UI automation tools like Appium and Calabash. Then we will discuss about the problems that have been created by the third party UI automation tools. And finally, we will talk about Apple's own UI testing framework, also known as XCUITest. And how we can use XCUITest for better iOS development.

So let's talk about a little bit of history of iOS development. So in the past, iOS teams usually very small having one or two iOS developer. And all the key activities had been handled by developers themselves. So there wasn't any need to have a dedicated QA team or QA person to test the applications. It may be because applications were not that complex and companies might not be earning that much revenues with their iOS apps.

But in recent days, the team has bene drastically changed. Now, businesses are adopting the Mobile First approach. And they have realized the importance of having iOS apps for their businesses. So as a result, they started hiring more and more iOS developers, so the team started growing. And that initiates need of having QA or QA team to test the iOS apps.

Although the team has grown in size, iOS developers has grown in size, there's a QA team as well, but the iOS development and testing practices still remaining unchanged. Although the technology has been [inaudible 00:02:14] a lot in terms of iOS like Apple has launched [inaudible 00:02:19] and the testing framework has been growing a lot, but the testing usually in iOS team hasn't grown a lot. Some companies still develop the app in silos and then hand it over to the QA team for testing. And for most of the teams, there's no CI/CD, TDD, BDD or any best Agile Development Practices implemented within iOS development.

So developer used to develop application using their local machines, and then build the application, and hand it to the QA team for testing. Then the QA used to test those things manually. And there's heavy manual testing involved during this phase. So the regression cycles might be a week or weeks or, in some cases, it would be months. And usually, it takes like months to release iOS apps to the users. So this approach is clearly not suitable for fast paced competitive world. As you might say the competitors might have good practices, agile development practices so that they can release the feature as soon as developed. So there might be a solution, some sort of solution to solve these problems.

So community has realized this is a problem with the testing in iOS world. And they badly need a tool, which is a Selenium like tool in iOS world. So they started developing the tools like Appium, Calabash, KIF, Frank, etc, and many more tools started coming into the market. So basically, these tools are developed using the Apple's native technologies like instrumentation or real world automation. And they take these technologies as a base and they write a Wrappers around those technologies. And they call themselves Mobile Testing Frameworks. But they are just Wrappers around Apple's native technologies.

So as QA started using those tools to avoid the heavy manual testing ... So instead of tapping on the simulator manually, these tools allows QA engineers to write some sort of script in languages like Ruby, Java, Python, C Sharp. And then QA engineers can write a script to tell simulators or, in some cases, real devices to tap on those screens. So instead of having the people tapping on the screen, now we have the scripts tapping the simulators. So now QA started spending more time on maintaining their tests. And they are no longer testing the real iOS tests, or they're not doing any explorative testing to find the issues, the real issues in iOS apps. So they spend time on maintaining their tests in Ruby, Java, and some other languages. And this basically created a huge technology gap because developers used to develop application in Swift or Objective-C. And the QA team started tracking their tests in Ruby, Java, Python, or some other crazy languages.

So one of the other things that the tools like Appium or Calabash got really popular and [inaudible 00:06:17] by most of the companies because of the cross platform support. Because we can write a test both for iOS and Android platform using those tools. And there's no need to have two different tests made. And we can get both platforms automated using those sort of tools. But to me, this approach in very dangerous. It can help to deliver the project on time, but any cross platforms solutions are never permanent. Either Apple or Google can break it one day. And you'll realize that the time, effort, and the money you put in on those cross platform frameworks are just gone in vain. And once these tools are [inaudible 00:07:08] by Apple or Google, then you probably have no other option to start from scratch again. So you have to be very, very careful about choosing those frameworks because you have to remember that the cross platform solution ... Any cross platforms solutions is just temporary hack that you are going to implement. And you are getting those hacks into your company or business as that's more dangerous than having this usually common [inaudible 00:07:41] approach.

So coming back the problems that has been created by UI automation frameworks. So the first one is a technology barrier. As we discussed earlier, the QA team is writing the script in Java, Ruby, or Python and the developer's writing the application coding in Swift or Objective-C. So the QA team has no idea how the application is being developed and the developer has no idea what's happening in that QA team. So this proves a huge technology barrier. Also, having those technical barriers, we can't put source code like a test code and the production code into the same repository because they are two

different technologies. So the most of the companies write development code or production code in one repository and put the test code in other repositories.

Implementing the continuous integration or continuous delivery practices within certain crazy technology differences is very hard. You can't just build in Swift and Java to get your continuous integration environment. So it's very hard to implement any agile development practices like CI/CD or TDD or BDD or we can't work in collaboration because of this technology barrier.

So another thing is the tests produced by those third party UI automation tools are very brittle. And they are usually slow because they are just a Wrapper. They need to first ... They can pass [inaudible 00:09:26] technologies and then they launch simulator and do the stuff. So tests are usually brittle and very slow. Another thing is those frameworks usually need a pre built app. So basically, we have to build an app in [inaudible 00:09:46] app format or in Appium format. And then we have to pass that app in the Appium. And then Appium or Calabash will take that app and then start testing that app. That means we can't work in parallel with development team. So we can't really start testing unless the developer finished this feature. So it's very hard to find that parallelism between development and testing. So it's, again, kind of [inaudible 00:10:21] technique that we have to wait for application of feature to be done and then we can start testing. So this approach is unsuitable as well.

So another thing is with those UI automation framework, Apple can break those tools any time. Whenever Apple change the API, these tools can break. I think that iOS 10 is a good example of that. So in iOS 10 release, Apple has duplicated the instrumentation technology. And all the third party tools like Appium, Calabash that build on those technologies has got broken because of that. They tried to get back with some hacks. I think some tools are already dead and some tools are still there just alive with some hacks. But these tools can be broken by Apple, or in case of Android, by Google, anytime.

Another thing. Whenever they, Apple, release new feature or new testing feature, they have to implement that feature into the Wrapper tools like Appium or Calabash. They're all [inaudible 00:11:31] with Apple's new feature, new testing features, and they need to implement them into those tools. And sometimes, it becomes very difficult to implement the different bindings. For example, someone can build it with a Java binding, but not in Ruby binding. So you know, it's always time consuming to get Apple's latest testing features into those frameworks. And we have to wait to get these things implemented in that Wrapper technologies, and then we can use into our project. So it is a very non process.

So we can't use the features that Apple has published straight away. Again, the silos, so developers and testers, can't work together with those tools. And all such kind of approaches like causing the damage to iOS development than good. So the whole approach of [inaudible 00:12:31] the automated simulators with those framework are not feasible.

So we have discussed a lot of things about problems with those third party UI automation tools. So one of the potential solution is Apple's own Xcode UI Testing framework, also known as XCUITest. This framework has been launched by Apple in 2015. You should definitely watch this [inaudible 00:13:00] on Xcode UI testing to understand this from scratch. So basically, XCUITest is a UI testing framework embedded within Xcode, which allows us to write a UI test for iOS apps in Swift or Objective-C. So one of the benefits of using XCUITest is it's way faster than all these Wrappers that we discussed before. So basically, it doesn't have to wait for anything. So it just build an application and start running the UI test. So it's way faster than the Wrapper tools.

Another thing is we don't have to wait until a feature is done. So we can take the code from any branch. We can build the target app. And we can start testing in parallel with development. Another thing is it's very easy to plug into the CI because we use the same technology testing framework from Apple called XCTest for unique testing as well as UI testing. So we can easily plug XCTest framework with our CI process. It makes the iOS developers happy because now developers understand what's going on in the QA team or what our application is really testing. They can contribute to the test code. They can add some value to the tests. And they can see the benefit of having [inaudible 00:14:32] to test development working process.

So there are lots of benefits XCUITest can produce within iOS 10 because if the developer is getting involved into those kind of processes, then it makes it very easy for the QA team to get some help from them, get more advice, make the framework more stable and solid by taking their help. However, if you try to go for XCUITest, you should remember some few things. We can't do the cross platform things in here because the XCUITest is only supposed to work with iOS applications. You can't do it for Android or some other platform. It's just for iOS. Another thing, I think potentially this is a major thing because the QA engineers need to learn Swift. And in most of the cases, QA engineers are mostly from Java background or C sharp background. And the world is full of the Java automation guys. You have to spend your time to learn Swift. And you have to make yourself familiar with the development platform and get involved with that.

Another thing is XCUITest is still new. And you might find some open bugs with that. But there are not major critical bugs. We can still use the functionality of XCUITest. But there are some minor things here and there floating around. But sooner or later, it's been fixed by Apple.

So why should you chose XCUITest? The main thing, it's maintained by Apple. So there's no chance that anyone can break these tools apart from Apple. Also, we can get the new features, new testing features straight into our workflow as Apple updates their tools. So we don't have to wait for someone to implement in the third party tools or something like that. So we can get these features straight away. Another thing is it speeds up the iOS development because we can collaborate with the developers and we can add more and more tests. We

can make a decision what to test or what not to test. And it increases the collaboration between QA and developers, which is I think one of the great advantages of having XCUITest.

And by using XCUITest, we are on the same page in terms of the technology. So we can easily adopt the best agile development practices like a TDD, BDD, continuous integration, and continuous delivery. And we can set up the release pipeline to just deploy the apps with automation and the practices like a TDD or BDD. But one of the major benefits of having XCUITest is a painless device testing. So in the world of Appium and Calabash, we need to have that pre made app and that app has to be signed by some provisioning profiles. And there's lots of cosigning activities involved in that process.

So with XCUITest, we can just attach the devices to our server. And if you have the certificates and profiles installed on the server machine, then it takes care of all these things. And since Xcode 8, these things become really painless because Apple has introduced automatic signing feature so that we don't have to worry about all these crazy certificates and provisioning profiles. So it has been all handled by the Xcode these days.

So we have talked a lot about XCUITest and it's benefit. So it's time to show you how it works in reality. So let's dive into the code and we'll see you in the demo.