Challenges in Mobile App Development

Brian Vanpee

Samer Fahmy

Quick Note

The views expressed in this presentation are solely our own and do not in any way represent those of our employers.

Agenda

- UI Challenges
- Technical Challenges
- QA Challenges

UI Challenges

Screen Resolutions

- Apps have to run on multiple different device types and resolutions
- Android devices for example ranges in size, resolution, dpi
- Some devices have keyboards with much smaller screens (e.g: BlackBerry Q10)
- Tablets larger screens that the developer would like to take advantage of
- An app can look quite different, and in many cases not work
- Impossible to test app on all variation of devices

Screen Resolutions

- Recommendation is to layout your UI dynamically
- Don’t rely on pixel values for layout, instead use constructs like “Center, Fill, Use Full Width” etc. where possible/available
- Keep a conscious split between your UI layout and your logic (e.g.: MVC)
  - This would allow easier porting to devices like tablets
- Replace specific layouts only where needed
  - To take advantage of keyboard for example

BB10 Cascades/QML Example

```
Container {
    layout: AbsoluteLayout {}

    Button {
        text: "Button"
        layoutProperties: AbsoluteLayoutProperties {
            positionX: 1820
            positionY: 500
        }
    }
}
```

```
Container {
    layout: DockLayout {}

    Button {
        text: "Button"
        horizontalAlignment: HorizontalAlignment.Center
        verticalAlignment: VerticalAlignment.Bottom
    }
}
```

BB10 Cascades/QML Example

```
Container {
    layout: AbsoluteLayout {}

    Button {
        text: "Button"
        layoutProperties: AbsoluteLayoutProperties {
            positionX: CONSTANTS.Button_X
            positionY: CONSTANTS.Button_Y
        }
    }
}
```

Screen Real-estate

- Very limited space on a Mobile Display, yet lots of information to show
- Discoverability is a challenge, how do you inform the user of all the features
- Menus can get cluttered with options, many that users don’t even find
- Gestures are great, but how does a user learn them?

Screen Real-estate

- Difficult problem to solve
- Try to not clutter the UI, but keep it simple, and introduce a flow to your UI that a user can follow
- Make main actions accessible and easily discoverable
- Use analytics to figure out what users use and don’t use, and bubble up those actions
- Visual hints are a good way to educate the user, however, can be intrusive if not done right

Simulators

- Of course you need to try running your code
- Multiple devices, versions, screens, it’s impossible to try them all
- Simulator is definitely a great option to develop and test your application
- Simulators don’t give the full picture however

Simulators

- Performance is not clearly visible on a simulator
- The feel of the application in the hand is different than on a monitor
- Can't represent hardware features like sensors well (e.g.: rotation, gyroscope)
- Pick a couple of candidates from the platforms you are targeting and run your application on the real devices
- For all the platforms you are targeting, use the lowest common denominator
  - If it runs well there, it will only be better on the higher end

Technical Challenges

Performance

- Performance is a huge concern on a Mobile platform
- Very limited resources, memory/CPU/GPU
- Users don’t sit in front of their smart phone for hours, they want quick access to information

Performance

- Start-up time, data loading time, UI Lag are some KPIs that are important
- You want the app to start up as quick as possible, and the app to respond (in some way) to a user action in times under 150ms
- Data loading should be quick
  - Spinners can mask the data loading, but can also make your app seem slow
- Load only what's absolutely necessary to start, and lazy-load the rest in the background

Performance

- There is always a tradeoff between memory and performance
- A key difference of mobile is an app takes up the entire screen
- Memory can be better leveraged by the running application
- Sometimes a better choice is to leverage memory to increase performance
  - Caching data if possible as an example

Battery Life

- Battery life is extremely important to users
- Every operation that is coded drains the battery
- You need to be mindful of battery consumption when you write your code
- Don’t run animations that aren’t necessary
- If you have to poll, be wise about when and how often you poll
- Network connections are high battery consumers, use only when you have to, try to batch your requests if you can

QA Challenges

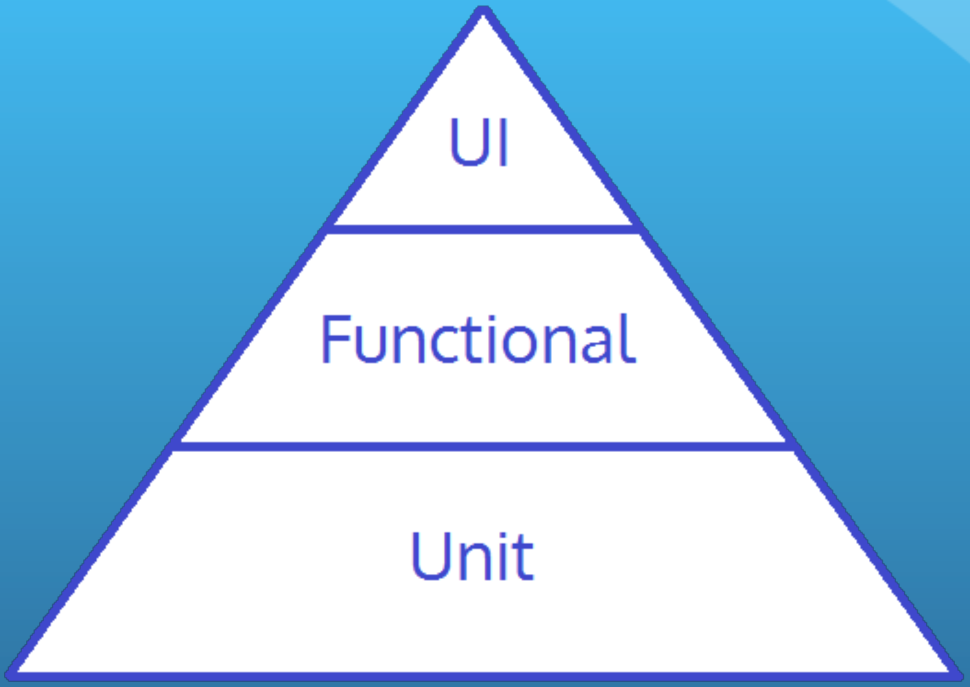
How to Test?

- Mobile platforms are immature
- Missing common testing tools
- Don’t run on common hardware
- Testing on real hardware can be expensive or difficult
- Simulators aren’t perfect

Options

- Skip Testing Altogether… (*gasp*)
- Raise an Army…
- Automation!
  - But How? First things first…

Back Up a Step



- Most tests are Unit tests
- Inverse time to run

Unit Tests

- Test Only One (Small) Thing!
- xUnit Frameworks
- Compile-Time or Run-Time?
- Really Off-Device or On-Device?

- Many teams choose to run Unit Tests On-Device…

Unit Tests

- On-Device has significant long-term costs:
  - Often become (very) slow over time
  - Require custom setups/hardware to run
  - Difficult to integrate into CI systems
  - Developers do not run them
- Off-Device at Compile-Time has none of these drawbacks
- So why choose On-Device?
  - The **Native/Library Problem…**

The Native/Library Problem

```
...
native void nativeCall();
...
import platformLibrary;
...
public class Foo {
    void doSomething() {
        platformLibrary.platformCall();
        // ...
        nativeCall();
    }
}
```

Solving The Native/Library Problem

- Introduce a little abstraction:

```
public class NativeWrapper {
    native void nativeCall();
}

public class Foo {
    void doSomething() {
        new NativeWrapper().nativeCall();
    }
}
```

Solving The Native/Library Problem

- Introduce some Dependency Injection:

```
public class Foo {
    Foo(NativeWrapper nativeWrapper) {
        this.nativeWrapper = nativeWrapper;
    }

    void doSomething() {
        nativeWrapper.nativeCall();
    }
}
```

Solving The Native/Library Problem

- Provides a way to mock-out platform specifics
- Convert run-time Unit Tests to compile-time
- Get around any platform limitation
- Can also use to remove:
  - Databases
  - Network
  - File I/O

Unit Tests

- Always prefer Off-Device tests executed at compile-time:
  - Faster
  - Easier to Run
  - Easier to Integrate into build/CI
  - Use existing xUnit/Mocking frameworks
  - Can be run from within IDE

Functional Tests

- Tests Several Parts Together
- Often ‘On-Device’:
  - Allowed to use ‘real’ Databases, File I/O, system services, etc.
  - Don’t have to be On-Device…

Easiest approach:

- Build app libraries/classes into a test app

UI Tests

- Test an App End-2-End as a User Would
- Must be ‘On-Device’
- *Very* Fragile
  - Highly dependent on UI design - names, layouts, screen ordering, etc.
  - High false-positive rate - many things even outside of app can go wrong
- … yet still worth doing!

UI Tests

- Good for ensuring basic functionality
- Most platform SDK's have this built-in:
  - uiautomator, Espresso for Android
  - KIF, Automation Instrument API for iOS
  - Truphone Labs for BlackBerry 10
- Will never fully replace manual testing
  - UI Testing tests static use cases, users are not static

UI Tests

Recommendations:

- Use for limited set of static use-cases
- Use when app is at or near completion
  - Great regression/stability testing of released versions undergoing maintenance
- Use in other cases where app (esp. UI) is not changing often/ significantly

Thank you for listening to us