Testing and Continuous Integration

Petar Šegina psegina@croz.net
Milan Jovanović mjovanovic@croz.net
Dejan Novak dnovak@croz.net

http://croz.net

Quality Assurance

- https://en.wikipedia.org/wiki/Quality_assurance
  - Quality assurance (QA) is a way of preventing mistakes or defects in manufactured products and avoiding problems when delivering solutions or services to customers

Why we test

- A bug-free app makes for a happy customer!
  - Certain things will be tolerated if the provided functionality is good enough
- Discovering and understanding bugs helps us avoid the same mistakes in the future
  - Write your code so that it’s easy to spot mistakes!
    - http://www.joelonsoftware.com/articles/Wrong.html

Testing – the standard way

- Use the app!
- Think about edge and corner cases!
- Write down your findings and steps to reproduce!
  - Use an issue tracker (e.g. GitLab)
  - A bug that happened once can happen again!
    - Make sure that the solution is well documented and easy to find!

Things to think of (1)

- Test your app in unusual environments!
  - No internet connection? **Bad** internet connection?
    - The emulator can simulate these!
  - Low battery? Low RAM? Doze? App Standby?
    - https://developer.android.com/training/monitoring-device-state/doze-standby.html
  - Scaled or windowed?

Things to think of (2)

- Test your app on unusual devices!
  - Different resolutions
  - Different screen sizes and ratios (1:1 ratio?!)
  - No Google Play services?
  - Various versions of Android
    - Various versions of various versions of Android
      - TouchWiz, Sense, ...

Things to think of (3)

- Test your app in unusual situations!
  - Call the device
  - Send a text message
  - Rotate the screen
    - Do not disable rotation in your app without a good reason!
- https://infinum.co/the-capsized-eight/articles/you-think-your-app-is-ready-for-publishing-well-think-again
  - *You think your app is ready for publishing? Well, think again.*

Developer options (1)

- Settings → System → About phone
  → Tap Build number many times
- Settings → System → Developer options
  - Take bug report + bug report shortcut
    - Better ways to do this (take care of your private data!)
  - USB debugging
  - Show touches + Pointer location (hardware test)
  - Show layout bounds
  - Force RTL direction

Developer options (2)

- Settings → System → Developer options
  - Transition and animation scales
  - Debug GPU overdraw
  - Simulate color space
  - **Strict mode**
    - https://developer.android.com/reference/android/os/StrictMode.html
  - **Don’t keep activities**

Android Studio debugger

- Stop on breakpoints
- Inspect the state of variables
- Move through code line by line
- Monitor logcat output
  - Write to logcat with Log.*
  - Be careful when writing sensitive data!

Monitoring tools

- Android Studio → Android Monitor
  - Run your app in debug mode (or attach to an existing process)
  - Monitor CPU, GPU, RAM and network
  - Profile CPU and GPU usage
  - http://c2.com/cgi/wiki?PrematureOptimization
    - "Programmers waste enormous amounts of time thinking about, or worrying about, the speed of noncritical parts of their programs, and these attempts at efficiency actually have a strong negative impact when debugging and maintenance are considered. **We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil. Yet we should not pass up our opportunities in that critical 3%.**"

adb – Android Debug Bridge

- adb devices
- adb logcat
  - pidcat, logcat-color
- adb shell
- adb install / uninstall
- adb shell screenrecord

Automated testing

- Testing by hand is
  - Time consuming
  - Error prone
- Can we improve this process?
- Some benefits of automated testing
  - Additional documentation
  - Avoiding regressions (This used to work!)
  - Peace of mind

Random testing with monkey

- adb shell monkey
  -p *package_name*
  -v *number_of_interactions*
- *https://developer.android.com/tools/help/monkey.html*
  - *You can use the Monkey to stress-test applications that you are developing, in a random yet repeatable manner.*
- If an exception occurs, you can reproduce the test with the given seed!

Scripted testing with monkeyrunner

- https://developer.android.com/tools/help/monkeyrunner_concepts.html
  - *With monkeyrunner, you can write a Python program that installs an Android application or test package, runs it, sends keystrokes to it, takes screenshots of its user interface, and stores screenshots on the workstation.*
  - *monkeyrunner can test application stability by running an application and comparing its output screenshots to a set of screenshots that are known to be correct.*

Introduction to Unit Testing

- Unit – a small, independent piece of code
- We test behavior, without worrying about implementation
- ./gradlew test
  - Gradle knows how to run our tests
  - Tests reports get generated
  - Exit code reflects success or failure

Unit Testing questions

- When?
  - Before the implementation?
  - During implementation?
  - After the implementation is already complete?
- How much?
  - What should we test?
  - Do we test all the cases?

Test driven development

1) Add a test
2) Run all tests and see if the new test fails
3) Write some code (minimum needed to pass the test - YAGNI)
4) Run tests
5) Refactor code

- https://en.wikipedia.org/wiki/Test-driven_development
- http://www.amazon.com/Test-Driven-Development-By-Example/dp/0321146530

Let’s test!

- What would the tests look like for a function
  int nextPrime(int number)?
- How do we write and run the test?

- testCompile 'junit:junit:4.11'
- testCompile 'org.easytesting:fest-assert:1.4'

Lessons learned

- Writing tests (usually) isn’t hard
- Tests execute much faster and save us time
- It’s easier to discover edge cases we might have not considered

- Do passing tests confirm that a project is bug-free?

Making code testable

- Sometimes our code may be hard to test due to too many dependencies
- Certain principles help us write more testable code
  - Architecture – MVP? MVVM?
    - Ted Mosby – Software Architect http://hannesdorfmann.com/android/mosby
  - Dependency Inversion and Dependency Injection
    - Easily mock dependencies with Mockito
      - http://mockito.org/

Writing Android UI tests (1)

- We'll use Espresso for testing the UI
  - https://developer.android.com/training/testing/ui-testing/espresso-testing.html
  - androidTestCompile 'com.android.support.test.espresso:espresso-core:2.2.1'
- Turn off transitions to avoid timing issues!
- Let's test a simple Hello, World! app

Writing Android UI tests (1)

- We'll use Espresso for testing the UI
  - https://developer.android.com/training/testing/ui-testing/espresso-testing.html
  - androidTestCompile 'com.android.support.test.espresso:espresso-core:2.2.1'
- Turn off transitions to avoid timing issues!
- Let's test a simple Hello, World! app

Writing Android UI tests (2)

- @RunWith(AndroidJUnit4.class)
- @LargeTest
- @Rule ActivityTestRule
- onView(withId()).perform()
- onView(withId()).check()

Automating test execution

- Git allows us to *hook* into certain tasks by writing scripts called hooks
- Gradle allows us to build and test the application from the command line (usually with a single command)
- What if we wrote a hook that ran the tests on a clean build every time code is pushed?
- If the tests succeed, allow the push (and deploy the app), otherwise report the errors

Continuous integration

- https://en.wikipedia.org/wiki/Continuous_integration
  - Continuous integration (CI) is the practice, in software engineering, of merging all developer working copies to a shared mainline several times a day.
- Various CI systems exist
  - GitLab CI, Travis CI, CircleCI, Jenkins
- Configured once
- A push is all that's needed to run the tests

Lecture 13 – Advanced topics

- Tell us what you want to hear about!
- Some examples:
  - Kotlin, Groovy, Retrofit, Dagger, RxJava, MVP, Push notifications, Android Wear, Android Auto, Bluetooth networking, animations and transitions, OpenGL, EventBus / Otto, ...

Homework

1. Test an app!

2. Write an implementation based on tests!

3. Write tests for your own app!