UNIT TESTING WORKSHOP

Android School
Mike Hurtado
@mikehc

HMO

ANDROID SCHOOL
NEARSOFT

WHO'S THAT GUY

- 3+ years of experience doing Android development.

- Experience in pretty big apps.

100,000 – 500,000

1M – 5M

5M – 10M

Number of installs

TESTING, WHAT'S THAT?

Software testing is an investigation conducted to provide stakeholders with information about the quality of the product or service under test.

Test techniques include, the process of executing a program or application with the intent of finding software bugs.

Software testing can also be stated as the process of validating and verifying that a software program/application/product:

1. meets the business and technical requirements that guided its design and development;
2. works as expected; and
3. can be implemented with the same characteristics.

Software testing, depending on the testing method employed, can be implemented at any time in the development process. However, most of the test effort occurs after the requirements have been defined and the coding process has been completed. As such, the methodology of the test is governed by the software development methodology adopted.

TESTING METHODS

BLACKBOX

Black box testing is a testing technique that ignores the internal mechanism of the system and focuses on the output generated against any input and execution of the system. It is also called functional testing.

WHITEBOX

White box testing is a testing technique that takes into account the internal mechanism of a system. It is also called structural testing and glass box testing.

TYPES OF TESTING

1. Unit Testing
2. Integration Testing
3. Functional Testing
4. System Testing
5. Stress Testing
6. Performance Testing
7. Usability Testing
8. Acceptance Testing
9. Regression Testing
10. Beta Testing

UNIT TESTING

Refers to tests that verify the functionality of a specific section of code, usually at the function level. In an object-oriented environment, this is usually at the class level, and the minimal unit tests include the constructors and destructors.

These type of tests are usually written by developers as they work on code (white-box style), to ensure that the specific function is working as expected. One function might have multiple tests, to catch corner cases or other branches in the code. Unit testing alone cannot verify the functionality of a piece of software, but rather is used to assure that the building blocks the software uses work independently of each other.

Unit testing is also called *component testing*.

UNIT TESTING - PROS

1. Find problems early
2. Facilitates change
3. Simplifies integration
4. Documentation
5. Design

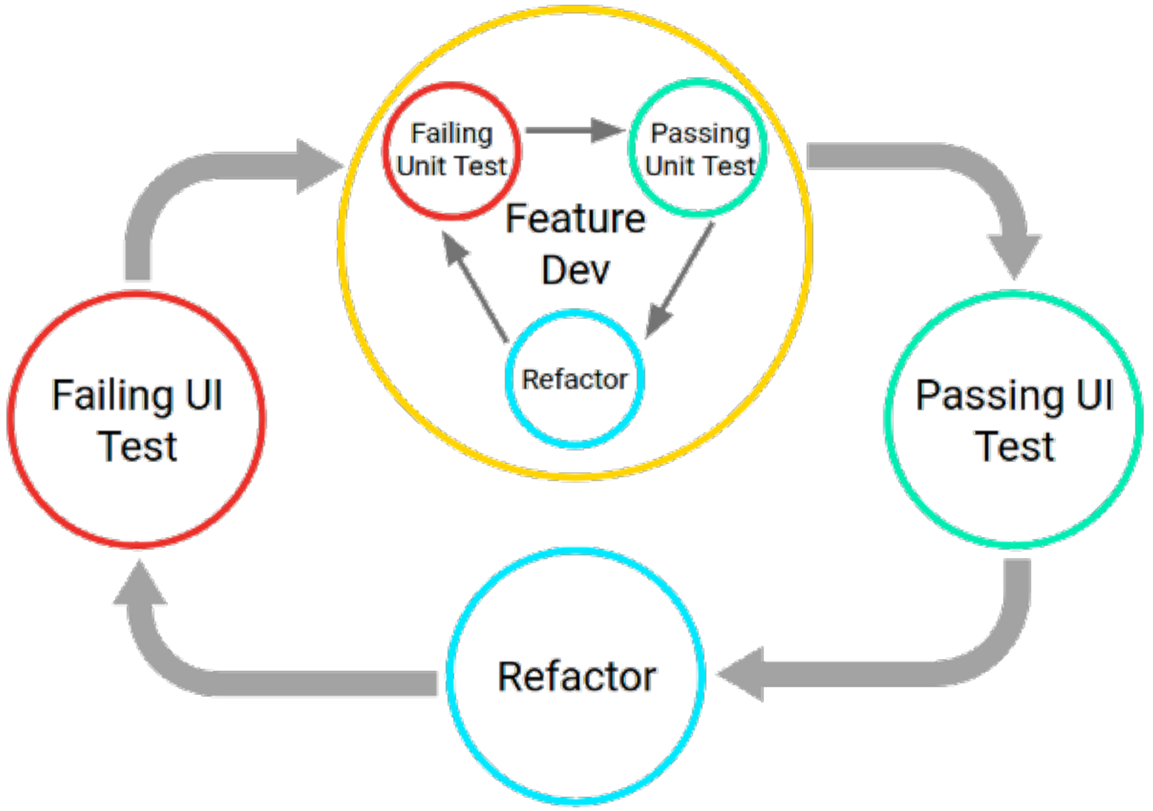
TEST DRIVEN DEVELOPMENT - TDD

Is a software development process that relies on the repetition of a very short development cycle: Requirements are turned into very specific test cases, then the software is improved to pass the new tests, only.

“TDD encourages simple designs and inspires confidence.” -Kent Beck

TESTS DRIVEN DEVELOPMENT

1. Add a test
2. Run all tests and see if the new test fails
3. Write the code
4. Run tests
5. Refactor code
6. Repeat



UNIT TESTING ON ANDROID

LOCAL TESTS

- Run on your local machine.

- Run on the Java Virtual Machine (JVM).
- Have no dependency on the Android Framework.

- Testing can be hard, need to use mock objects to fill gap.

INSTRUMENTED TESTS

- Run on an Android device or emulator.

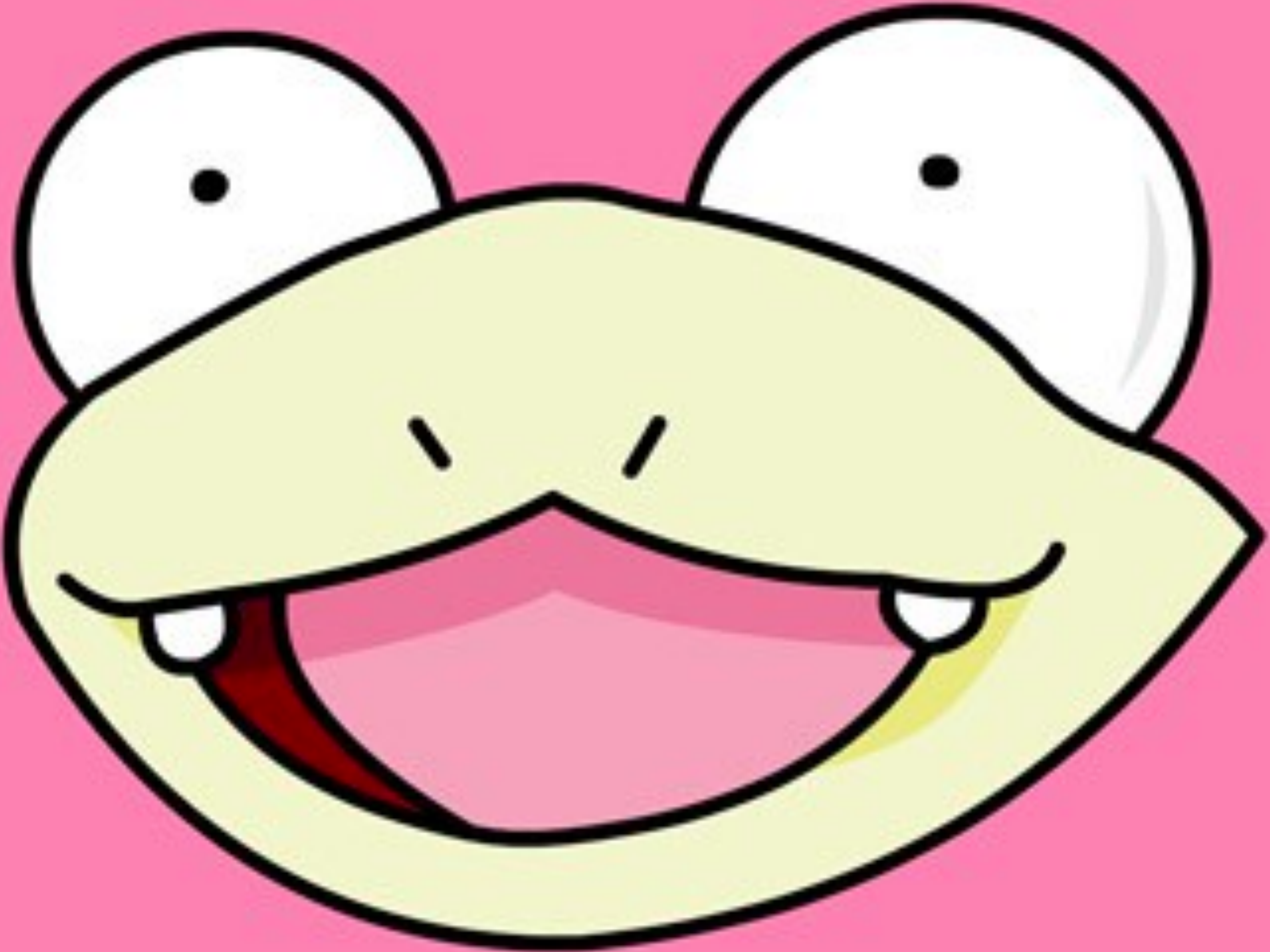
- They are slow.

- These tests have access to instrumentation information, such as the Context for the app under test.

- Use this approach to run unit tests that have Android dependencies which cannot be easily filled by using mock objects.

- SLOW!

SLOOOOOOOOOW



UNIT TESTING ON ANDROID

UNIT TESTS (METHOD)

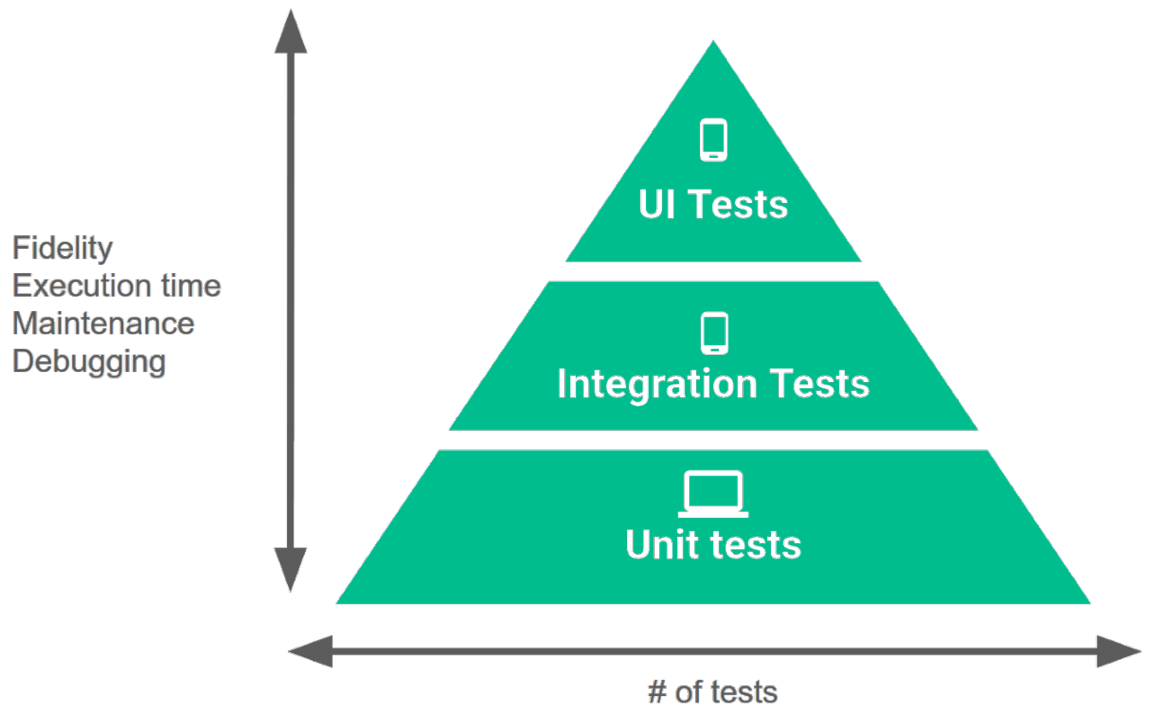
Can run insolated, they don't need other systems. Can run directly on your machine and they run FAST!

INTEGRATION TESTS (PROCESSES)

Need integration between other components of the app (systems, hardware, etc.) They may need to run on emulators or real devices.

UI TESTES (ACTIVITY)

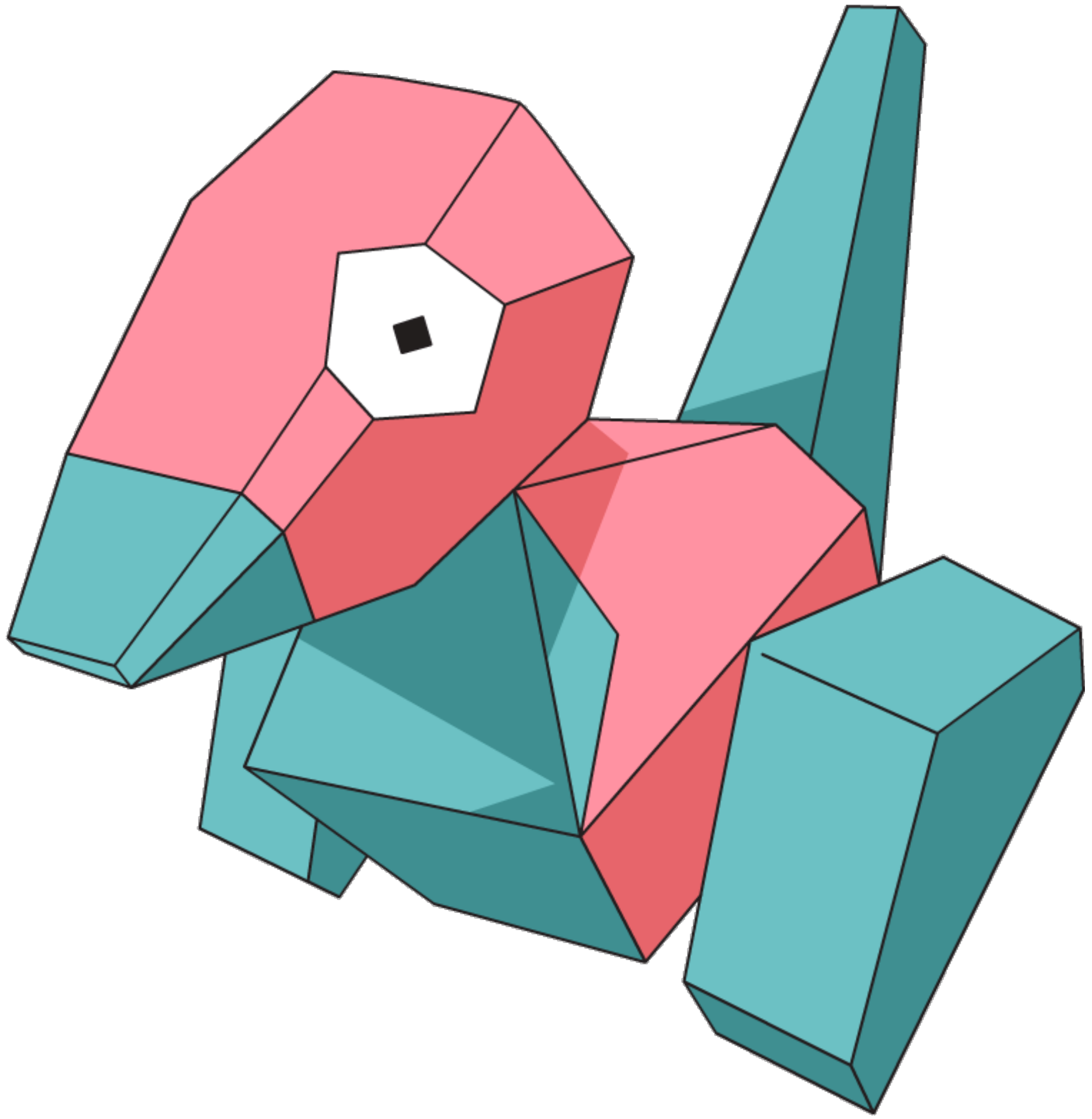
Integration tests that run by completing a UI workflow. The key is to ensure the user experience is the expected on emulators or real devices.



TESTING FRAMEWORK

There are a plethora of testing frameworks out there but we like to use these ones for Android

1. JUnit
2. Mockito
3. Power Mockito
4. Robolectric
5. Espresso
6. Hamcrest
7. *And more…*



EXERCISE

Android School

APP: NOT TOMATORO

NOT TOMATORO is a helper app for the Pomodoro technique.

The **Pomodoro Technique** is a time management method developed by Francesco Cirillo in the late 1980s. The technique uses a timer to break down work into intervals, traditionally 25 minutes in length, separated by short breaks. These intervals are named ***pomodoros***, the plural in English of the Italian word *pomodoro* (tomato), after the tomato-shaped kitchen timer that Cirillo used as a university student.

NOT TOMATORO

FEATURES

- Has a timer with two present intervals (pomodoros) for breaks and one for work.
- Implements some features of Material Design.

HOW IT WORKS?

- Select the type of pomodoro.
- Press play.
- ???
- Profit.

NOT TOMATORO

TECH STACK

- KOTLIN
- Unit test framework
- Bullet MVC*

CODE

```
git clone https://github.com/Nearsoft/android-school.git
cd android-school/2017/oct/unit-testing/PomodoroKotlin
git checkout clean_exercise
```

* Bad English

EXERCISE #1

CREATE A TESTS FOR ALREADY IMPLEMENTED FUNCTIONS

1. We want to be sure our presenter is working as intended. See

   TimerPresenterTest.kt

2. Complete the tests based on the implementation of the *presenter*.

3. Don’t forget to check also the *view*.

EXERCISE #2

VERIFY A BUG FIX

For some reason when we open the app for the first time the text of the timer is blank. We need to fix that.

1. Fix the bug.
2. Write a test in *TimerFragmentTest.kt*

EXERCISE #3

IMPLEMENT A FEATURE BASED ON TESTS/SPECS

We want the app to have a better UX. The app already allows the user to play, pause and resume pomodoros. We need to reflect to the user the changes in states of the app.

1. Change the UI code to implement the spec.

HMO

ANDROID SCHOOL

NEARSOFT

THANKS!

Mike Hurtado
mhurtado@nearsoft.com
@mikehc