# Kotlin Test Robots

ashdavies.io

# Testing


ashdavies.io

# Tooling

ashdavies.io


ashdavies.io

# UI Testing

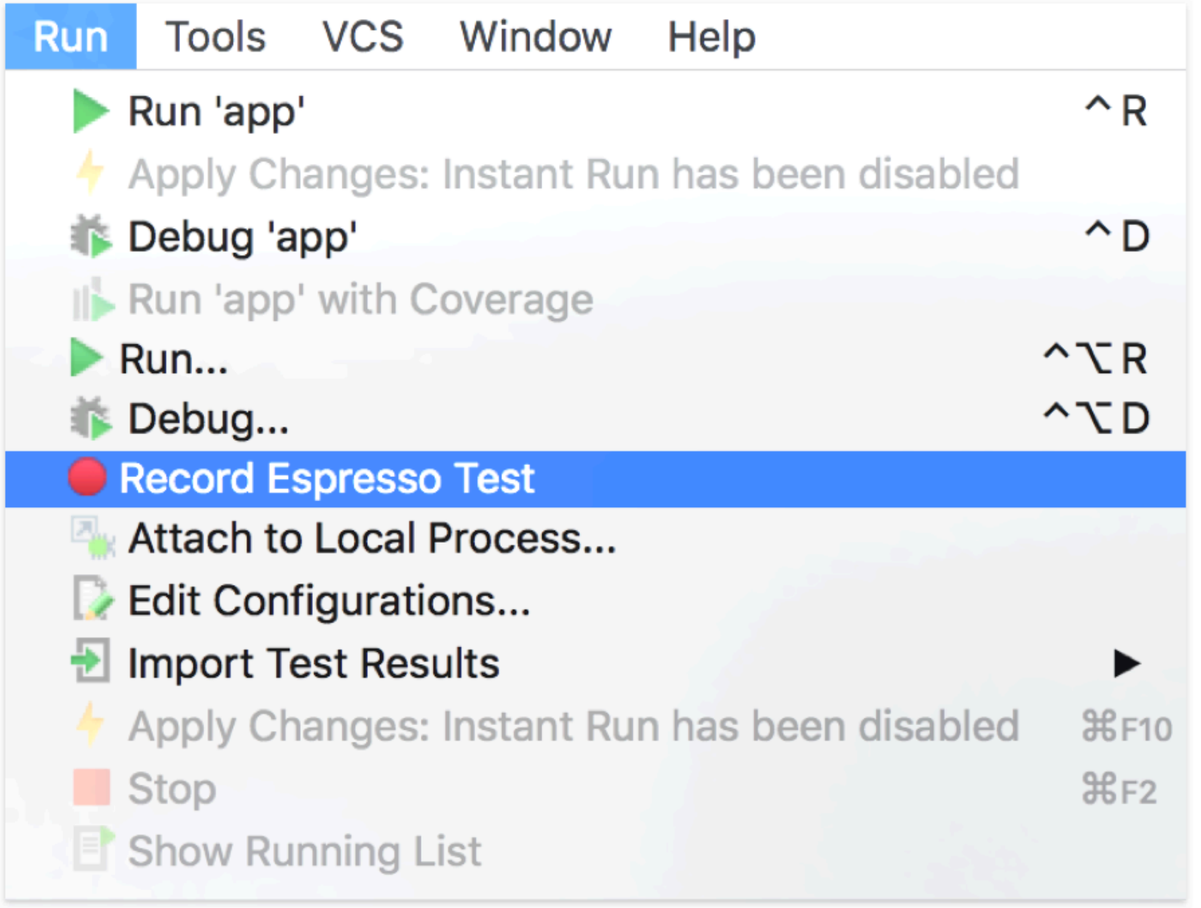

ashdavies.io

# Architecture Agnostic

ashdavies.io

# Espresso

ashdavies.io




# Espresso Test Recorder

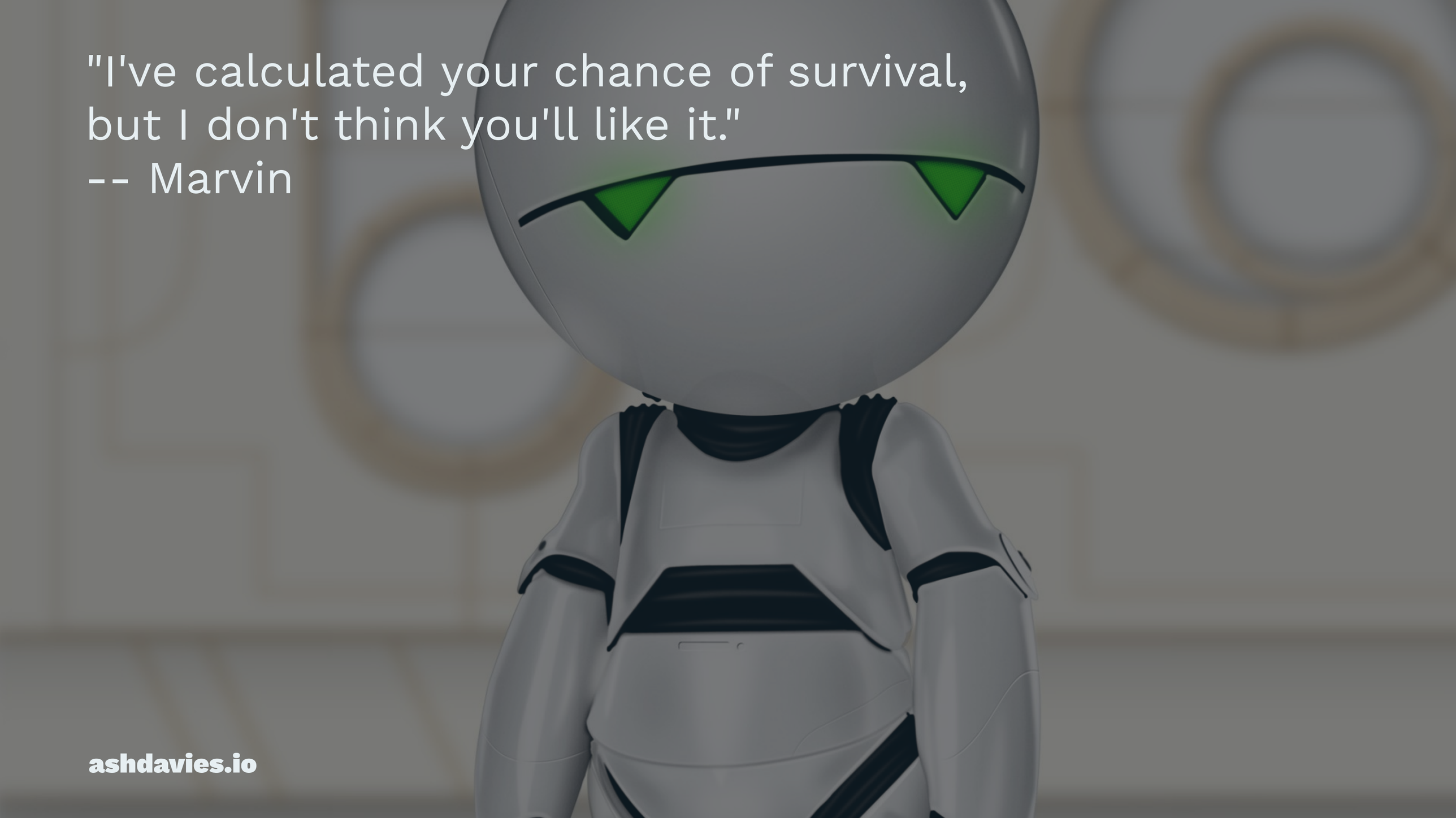
## Recording an Espresso Test

```
ViewInteraction appCompatEditText = onView(
        allOf(withId(R.id.email),
                withParent(allOf(withId(R.id.activity_main),
                        withParent(withId(android.R.id.content)))),
                isDisplayed()));
appCompatEditText.perform(click());

ViewInteraction appCompatEditText2 = onView(
        allOf(withId(R.id.email),
                withParent(allOf(withId(R.id.activity_main),
                        withParent(withId(android.R.id.content)))),
                isDisplayed()));
appCompatEditText2.perform(replaceText("test@invalid"), closeSoftKeyboard());

ViewInteraction appCompatButton = onView(
        allOf(withId(R.id.validate_button), withText("Validate"),
                withParent(allOf(withId(R.id.activity_main),
                        withParent(withId(android.R.id.content)))),
                isDisplayed()));
appCompatButton.perform(click());

ViewInteraction textView = onView(
        allOf(withId(R.id.validation_result), withText("Invalid Email Format"),
                childAtPosition(
                        allOf(withId(R.id.activity_main),
                                childAtPosition(
                                        withId(android.R.id.content),
                                        0)),
                        1),
                isDisplayed()));
textView.check(matches(isDisplayed()));
```


"I've calculated your chance of survival,
but I don't think you'll like it."
-- Marvin
ashdavies.io

# Espresso

ashdavies.io

# Espresso

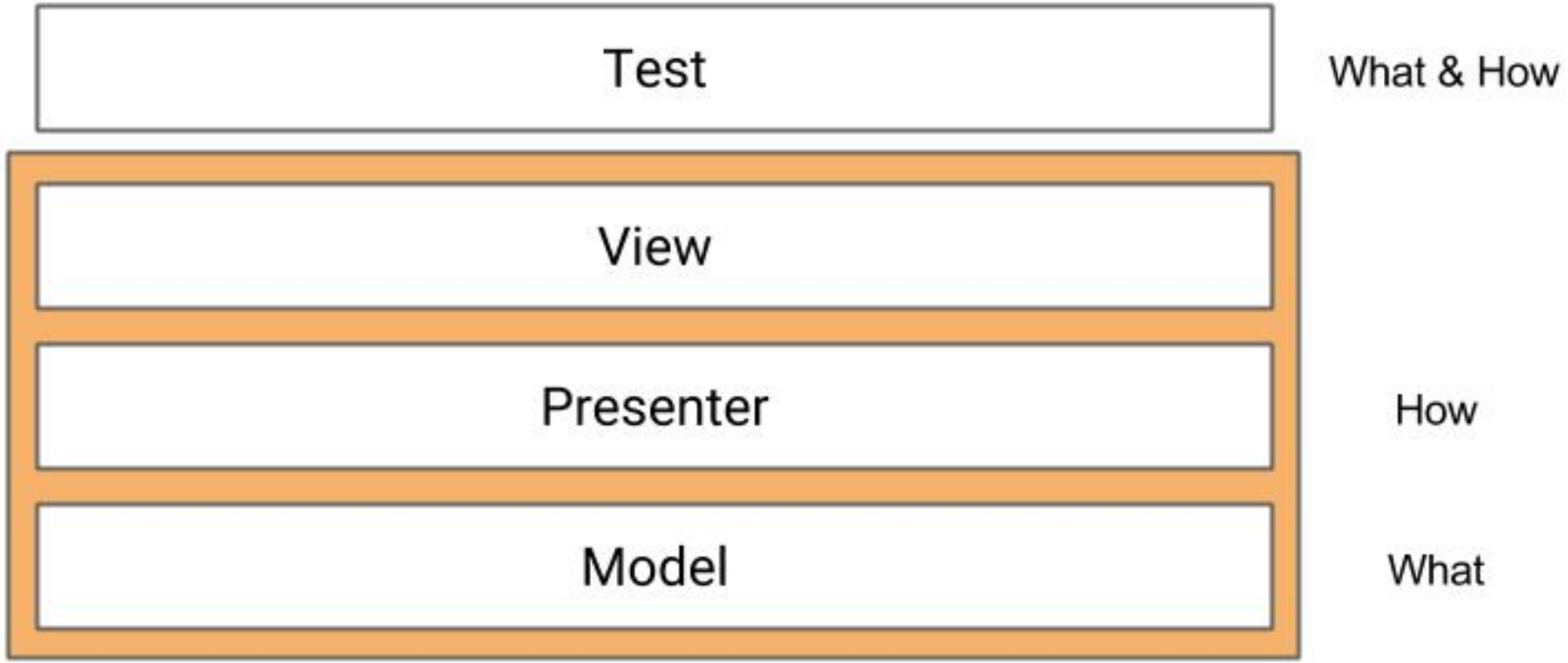
```
@Test
fun shouldValidateEmailAddress() {
    onView(withId(R.id.email)).check(matches(isDisplayed()))
    onView(withId(R.id.email)).perform(typeText("ash"))

    onView(withId(R.id.login)).perform(click())
    onView(withId(R.id.error)).check(matches(allOf(isDisplayed(), withText(R.string.invalid_email))))
}
```

# Espresso

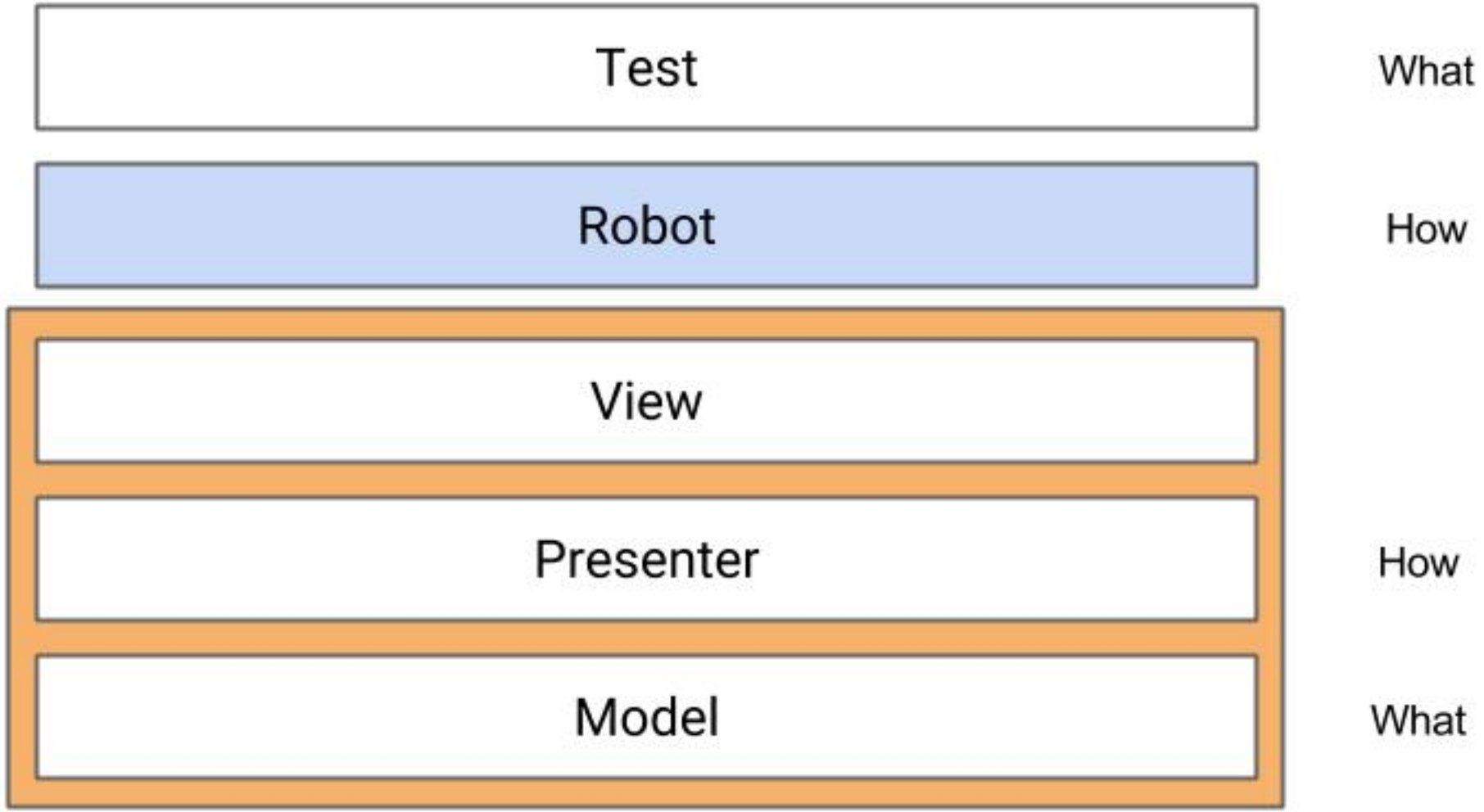
```
@Test
fun `should validate email address`() {
    onView(withId(R.id.email)).check(matches(isDisplayed()))
    onView(withId(R.id.email)).perform(typeText("ash"))

    onView(withId(R.id.login)).perform(click())
    onView(withId(R.id.error)).check(matches(allOf(isDisplayed(), withText(R.string.invalid_email))))
}
```



# Robots

ashdavies.io



# Robots

# Robots

— Structure your own domain language

# Robots

— Structure your own domain language
— Contextualise assertions and verifications

# Robots

— Structure your own domain language
— Contextualise assertions and verifications
— High level intention language

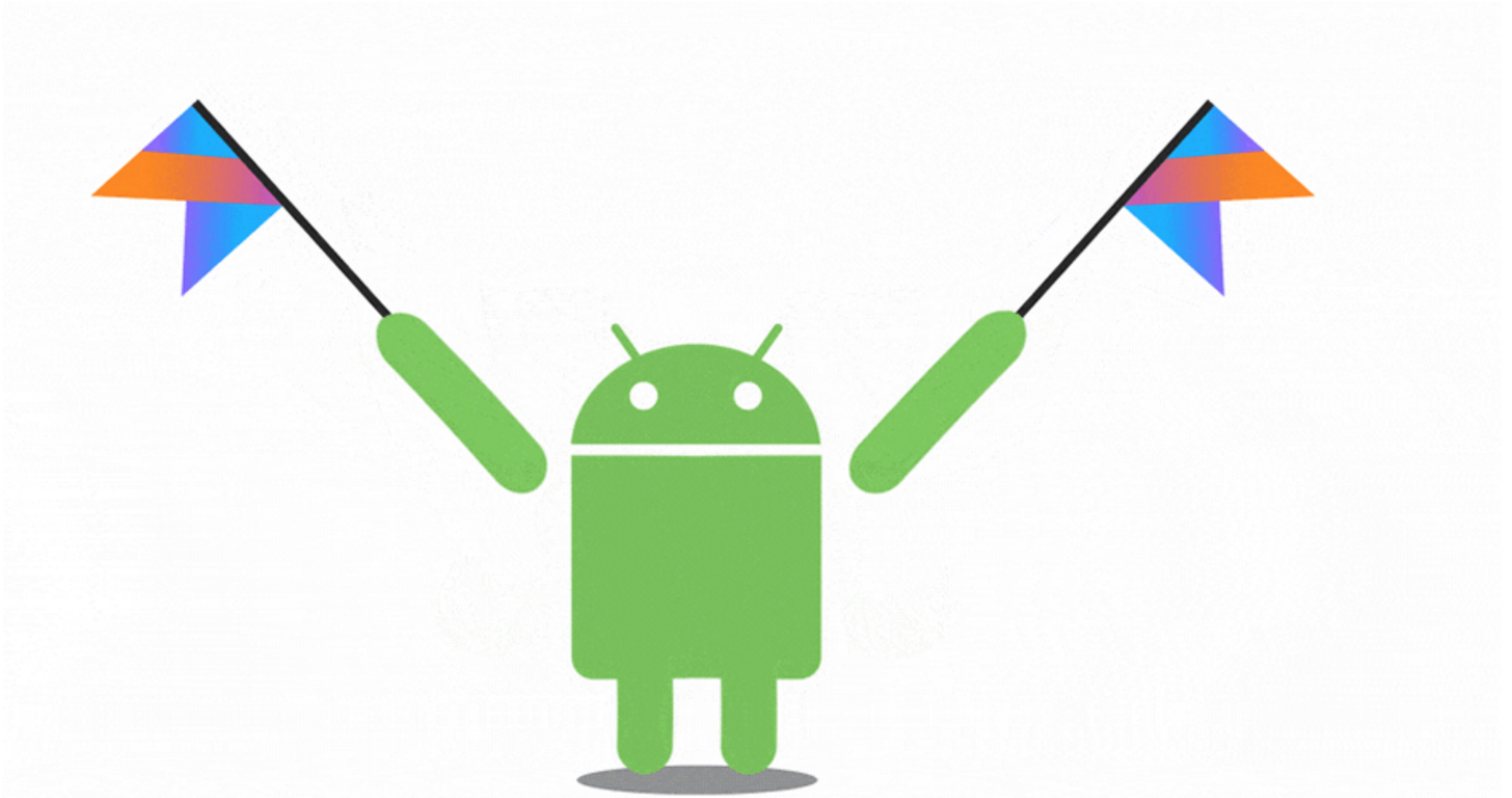
# Robots

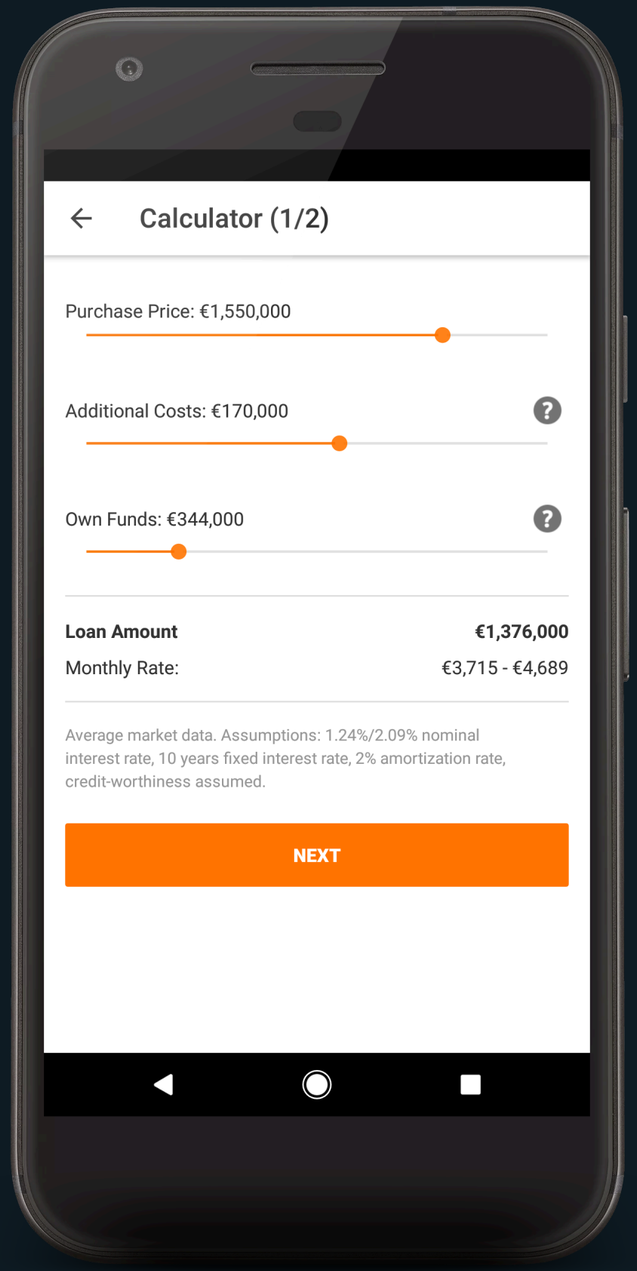
— Structure your own domain language
— Contextualise assertions and verifications
— High level intention language
— Reads like a script

# Sound Familiar?

## Testing Robots (Kotlin Night) - JW

ashdavies.io






ashdavies.io

# Robot

```
class FinanceRobot {

    fun price(amount: Int): FinanceRobot { ... }

    fun costs(amount: Int): FinanceRobot { ... }

    fun funds(amount: Int): FinanceRobot { ... }

    fun next(): ResultRobot { ... }
}
```

# Robots

```
class FinanceRobot {

    fun price(amount: Int) = apply {
        onChild(withId(R.id.price)).perform(typeText(value))
    }

    fun costs(amount: Int) = apply {
        onChild(withId(R.id.costs)).perform(typeText(value))
    }

    fun funds(amount: Int) = apply {
        onChild(withId(R.id.funds)).perform(typeText(value))
    }

    fun next() = apply {
        onChild(withId(R.id.next)).perform(click())
    }
}
```

```
class FinanceTest {

    fun `should update seek bar amount`() {
        FinanceRobot()
            .price(200_000)
            .next()
    }
}
```

# Cucumber

## Gherkin

```
Feature: Some terse yet descriptive text of what is desired
   Textual description of the business value of this feature
   Business rules that govern the scope of the feature
   Any additional information that will make the feature easier to understand

   Scenario: Some determinable business situation
       Given some precondition
           And some other precondition
       When some action by the actor
           And some other action
           And yet another action
       Then some testable outcome is achieved
           And something else we can check happens too
```

# Kotlin

```kotlin
@Test
fun `should update progress bar amount`() {
    FinanceRobot()
        .given { amount(200_000) }
        .then { progress(200_000) }
}
```

# Kotlin

```
class FinanceRobot {

    fun given(func: Given.() -> Unit) = Given().apply(func)

    class Given {

        fun amount(value: Int) = apply {
            onChild(withId(R.id.input)).perform(typeText(value))
        }
    }
}
```

# Kotlin

```
class FinanceRobot {

    fun given(func: Given.() -> Unit) = Given().apply(func)

    class Given {

        fun amount(value: Int) = apply {
            onChild(withId(R.id.input)).perform(typeText(value))
        }
    }
}
```

# Kotlin

```
class FinanceRobot {

    fun given(func: Given.() -> Unit) = Given().apply(func)

    class Given {

        fun amount(value: Int) = apply {
            onChild(withId(R.id.input)).perform(typeText(value))
        }
    }
}
```

# Kotlin

```
class FinanceRobot {

    fun given(func: Given.() -> Unit) = Given().apply(func)

    class Given {

        fun amount(value: Int) = apply {
            onChild(withId(R.id.input)).perform(typeText(value))
        }

        fun then(func: Then.() -> Unit) = Then().apply(func)
    }
}
```

# Kotlin

```
class FinanceRobot {

    fun given(func: Given.() -> Unit) = Given().apply(func)

    class Given {

        fun inputText(value: Int) = apply {
            onChild(withId(R.id.input)).perform(typeText(value))
        }

        fun then(func: Then.() -> Unit) = Then().apply(func)
    }

    class Then {

        fun hasProgress(value: Int) = apply {
            onChild(withId(R.id.progress)).perform(scrubProgress(value))
        }
    }
}
```

# IntentsTestRule

```
@get:Rule
val rule: IntentsTestRule<FinanceActivity> = FinanceRobot.rule(FinanceActivity::class.java, false, false)
```

# Robot Companion

```
class FinanceRobot {

    companion object {

        fun start(rule: IntentsTestRule<FinanceActivity>, amount: Int = 200_000): FinanceRobot {
            rule.launchActivity(FinanceActivity.newIntent(context, amount))
            return FinanceRobot()
        }
    }
}
```

# Testing Robots

```
@Test
fun `should update progress bar amount`() {
    FinanceRobot.start(rule)
        given { inputText("200000") }
        then { hasProgress(200_000) }
}
```

# Testing Robots

```
@Test
fun `should update progress bar amount`() {
    FinanceRobot.start(rule)
        given { /* ??? */ }
        then { /* ??? */ }
}
```

# Kotlin

```
class FinanceRobot {

    fun given(func: Given.() -> Unit) = Given().apply(func)

    class Given {

        fun inputText(value: Int) = apply { /* ... */ }

        fun then(func: Then.() -> Unit) = Then().apply(func)
    }

    class Then {

        fun hasProgress(value: Int) = apply { /* ... */ }
    }
}
```

```
class FinanceRobot {

    fun given(func: Given.() -> Unit) = Given().apply(func)

    class Given {

        fun price(func: Will.() -> Unit) = Will(R.id.price).apply(func)

        class Will(@IdRes parent: Int) {

            fun inputText(value: Int) = apply { /* ... */ }

            fun then(func: Then.() -> Unit) = Then().apply(func)
        }
    }

    class Then {

        fun hasProgress(value: Int) = apply { /* ... */ }
    }
}
```

```
class FinanceRobot {

    fun given(func: Given.() -> Unit) = Given().apply(func)

    class Given {

        fun price(func: Will.() -> Unit) = Will(R.id.price).apply(func)

        class Will(@IdRes parent: Int) {

            fun inputText(value: Int) = apply {
                onView(allOf(isDescendantOfA(withId(parent)), withId(R.id.input)))
                    .perform(typeText(value)).perform(pressImeActionButton())
                }

            fun then(func: Then.() -> Unit) = Then().apply(func)
        }
    }

    class Then {

        fun hasProgress(value: Int) = apply { /* ... */ }
    }
}
```

```
class FinanceRobot {

    fun given(func: Given.() -> Unit) = Given().apply(func)

    class Given {

        fun price(func: Will.() -> Unit) = {
            Will(R.id.price).apply(func)
            return FinanceRobot()
        }

        class Will(@IdRes parent: Int) {

            fun inputText(value: Int) = apply { /* ... */ }
        }
    }

    fun then(func: Then.() -> Unit) = Then().apply(func)

    class Then {

        fun hasProgress(value: Int) = apply { /* ... */ }
    }
}
```

# Testing Robots

```
fun `should update progress bar amount`() {
    FinanceRobot.start(rule)
        given {
            price { inputText("200000") }
            funds { scrubTo(40_000) }
        }
        then {
            price { hasText("200.000 €") }
            funds { hasText("40.000 € / 20 %") }
        }
}
```

# Testing Robots

```
fun `should update progress bar amount`() {
    rule.start()
        given {
            price { inputText("200000") }
            funds { scrubTo(40_000) }
        }
        then {
            price { hasText("200.000 €") }
            funds { hasText("40.000 € / 20 %") }
        }
}

fun IntentsTestRule<FinanceActivity>.start(amount: Int = 200_000): FinanceRobot {
    return FinanceRobot.start(this, amount)
}
```

# Thanks

ashdavies.io