Testing

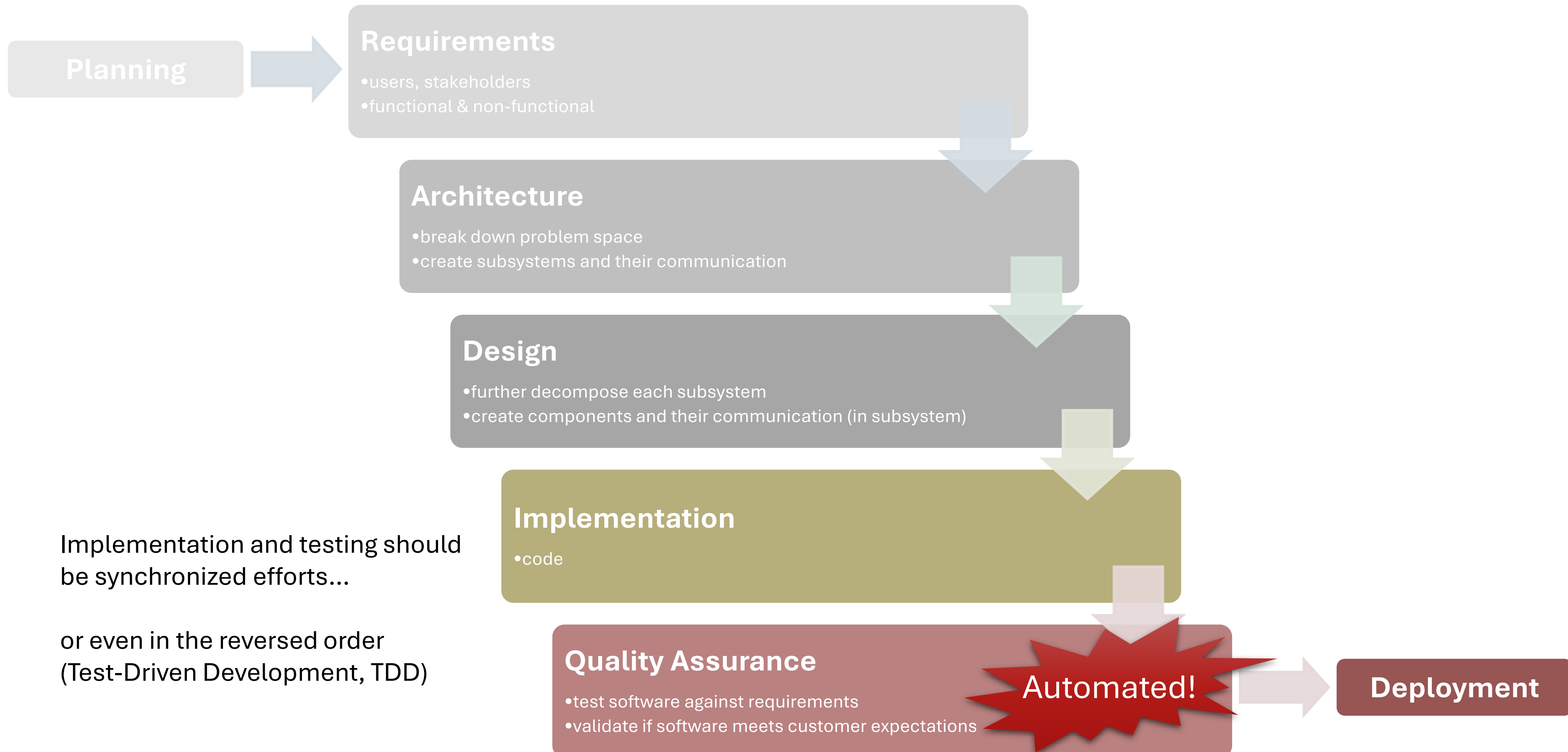
Pengyu Nie

Acknowledgements: slides adapted from previous versions by Mei Nagappan and Shane McIntosh, which are adapted from previous versions by Zhen Ming Jiang, Ahmed E. Hassan, Reid Holmes.

Agenda

- Testing basics
- Categories
  - by granularity
  - by subject
  - by methodology
- Testing libraries and techniques (w/ Demo)
  - running
  - measuring coverage

Revisiting Software Development Lifecycle



Implementation and testing should be synchronized efforts...

or even in the reversed order (Test-Driven Development, TDD)

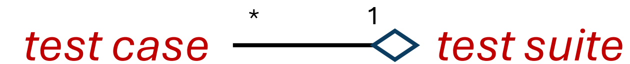
What is Test and Why?

- “The process of evaluating and verifying that a software product or application does what it’s supposed to do” (*requirements*)
- Why
  - Prevent bugs (from troubling users of software)
  - Ensure software quality
  - Improve performance
- Takes ~50% of software development time!

Source: https://www.ibm.com/topics/software-testing

Arrange, Act, Assert

code under test




```kotlin
class TipCalculator {
    var amount: Double = 0.0
    var tipPercent: Double = 0.0
    var roundUp: Boolean = false

    fun calculateTip(): Double {
        var tip = tipPercent / 100 * amount
        if (roundUp) {
            tip = ceil(tip)
        }
        return tip
    }
}
```

name

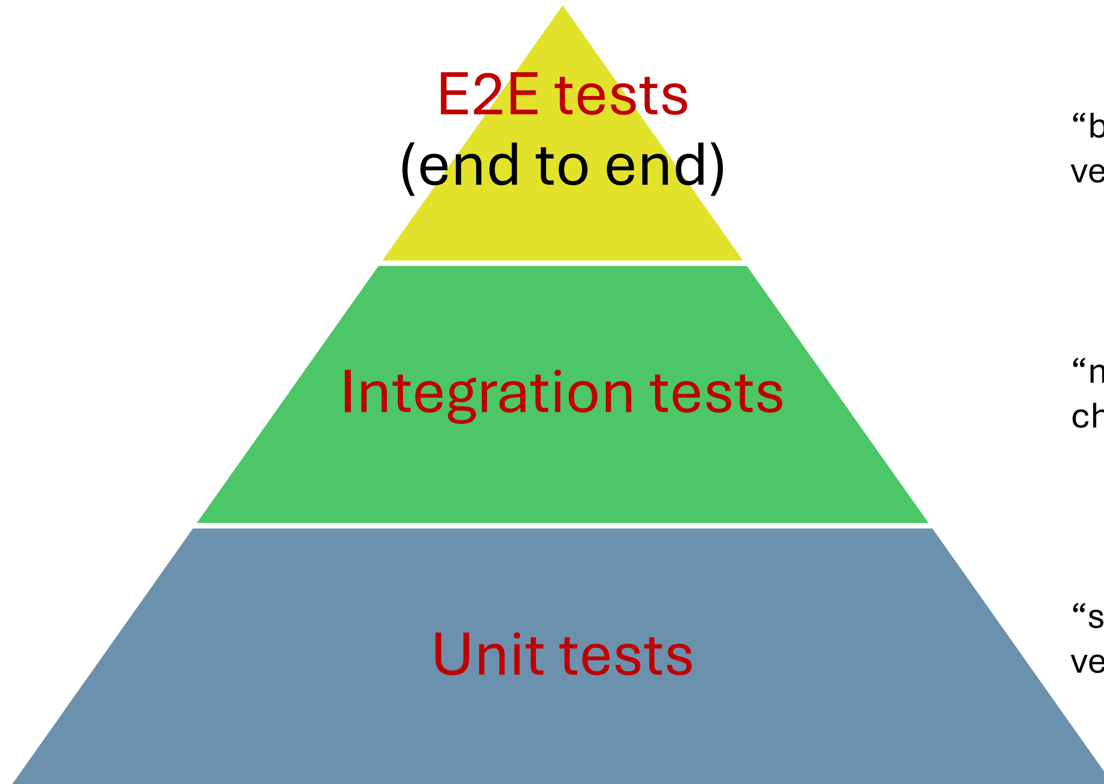
arrange
prepare test inputs

act
invoke code under test

assert aka *oracles*
check expected outcomes

```kotlin
@Test
fun testCalculateTip() {
    val calculator = TipCalculator()
    calculator.amount = 42.0
    calculator.tipPercent = 10.0

    val tip = calculator.calculateTip()

    assertEquals( expected: 4.2, tip,  delta: 1e-6)
}
```

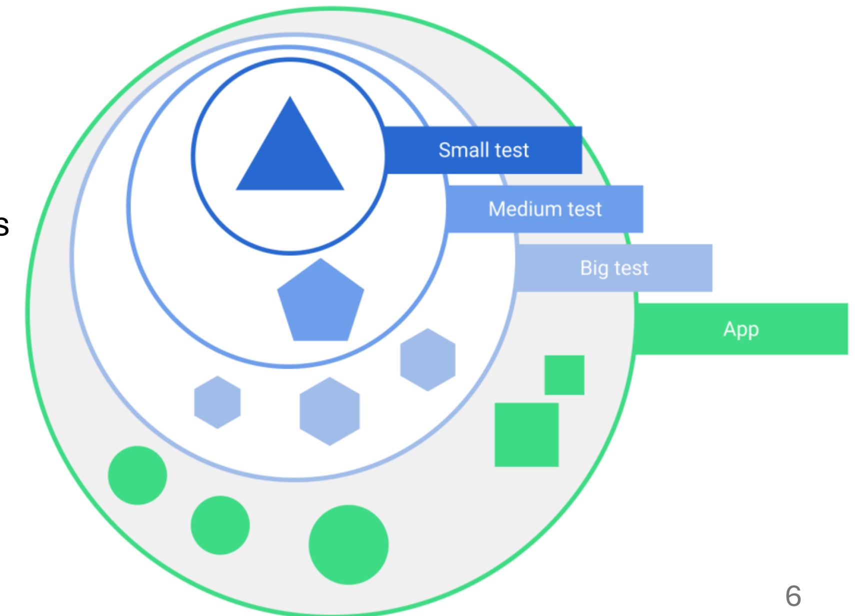
Tests by Granularity



"big tests"
verifying a end-to-end workflow (user scenario)

"medium tests"
checking the integration between several units

"small tests"
verifying a method or class

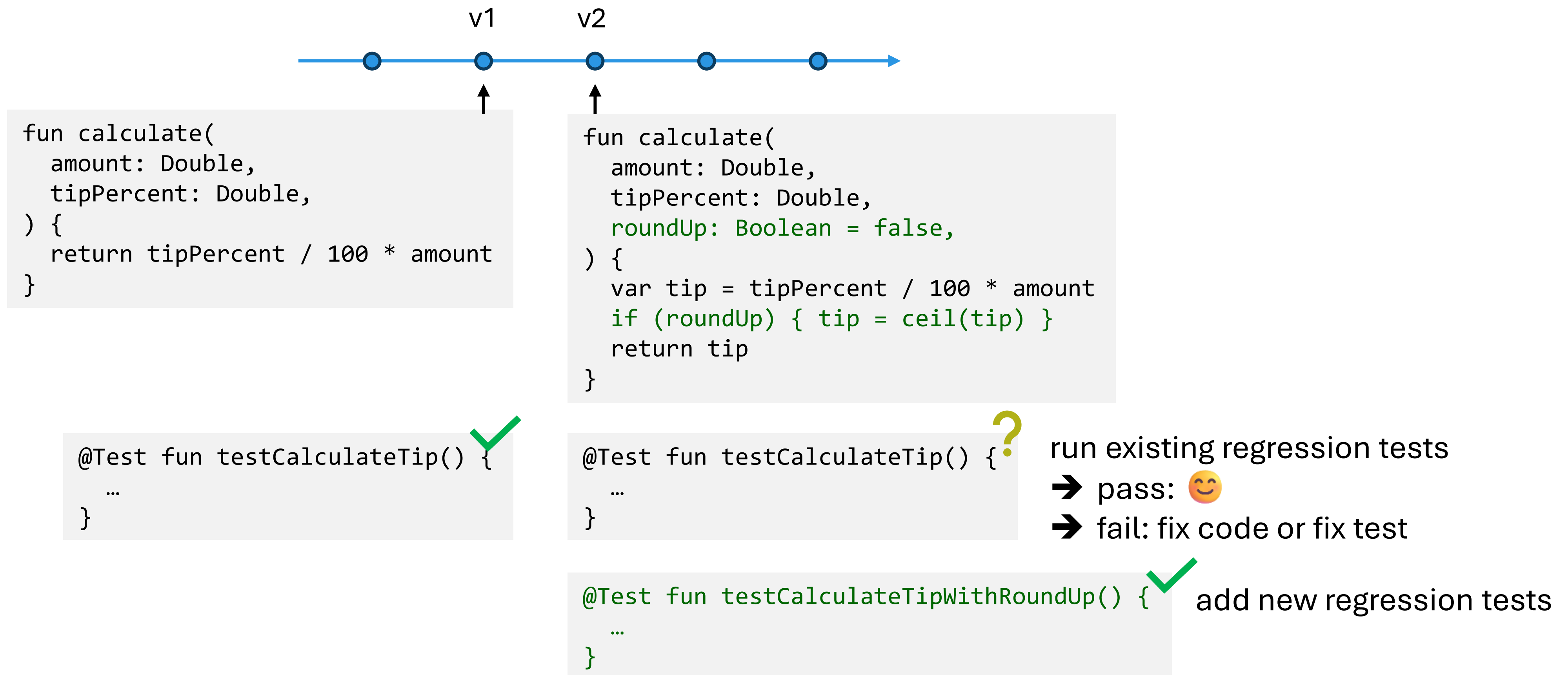


Tests by Subject

- Functional tests
  - focus on the business logic and functional requirements
- UI tests
  - focus on the user interface, usually integration/E2E tests
- Performance tests
  - focus on checking if code runs efficiently
- Accessibility tests, Compatibility tests, etc.

Tests by Methodology: Regression Tests

- Make sure your existing functionalities are not broken after code changes

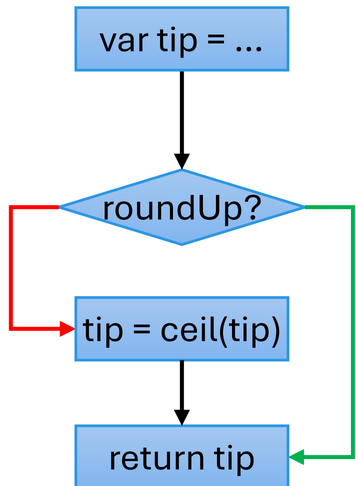


Tests by Methodology

- Regression tests
  - protect existing functionalities from being broken by code changes
- Random tests (fuzzing, monkey tests)
  - explore diverse execution paths of the software
  - test oracle: not crashing / not violating invariants
- Differential tests
  - compare the execution results when running on different devices
  - test oracle: they should be the same
- Metamorphic tests
  - check the relationships of the outputs when giving a set of related inputs
  - test oracle: $P_{precond}(x_1, x_2, \ldots) \rightarrow P_{postcond}(y_1, y_2, \ldots)$

Code Coverage

- Quality metric of your test suites
  - target 80-90%, if not 100%
- What % of code elements is “covered” (executed) during tests?
  - line coverage 3 / 4 lines = 75%
  - branch coverage 1 / 2 branches = 50%



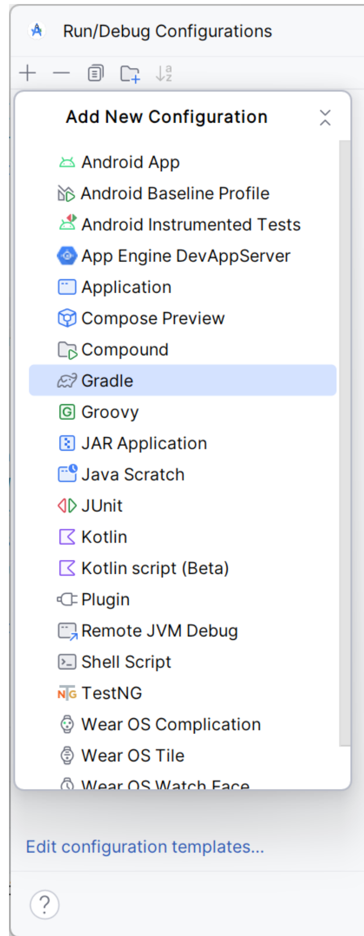

```
class TipCalculator {
    var amount: Double = 0.0
    var tipPercent: Double = 0.0
    var roundUp: Boolean = false

    fun calculateTip(): Double {
        var tip = tipPercent / 100 * amount
        if (roundUp) {
            tip = ceil(tip)
        }
        return tip
    }
}
```

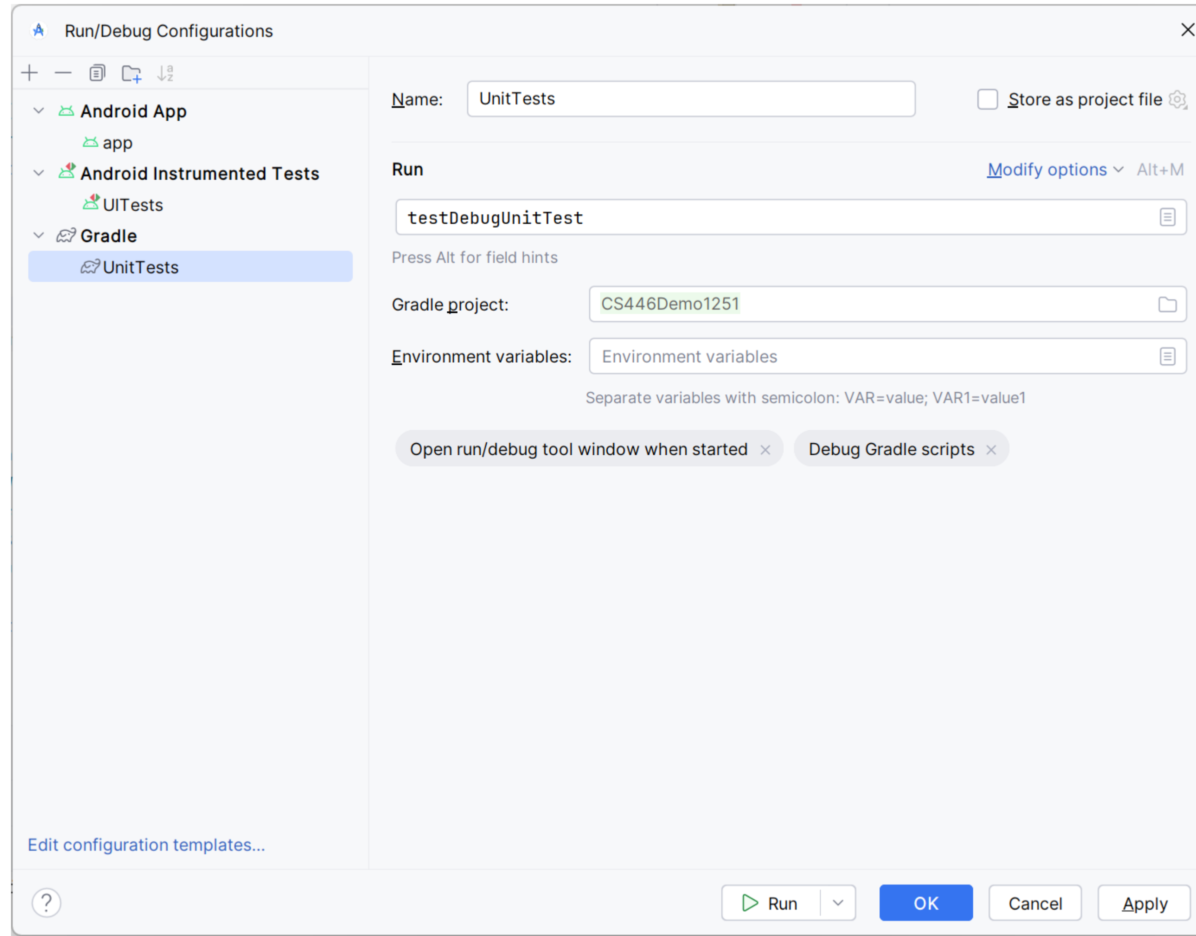
```
@Test
fun testCalculateTip() {
    val calculator = TipCalculator()
    calculator.amount = 42.0
    calculator.tipPercent = 10.0

    val tip = calculator.calculateTip()

    assertEquals( expected: 4.2, tip, delta: 1e-6)
}
```

Testing in Android

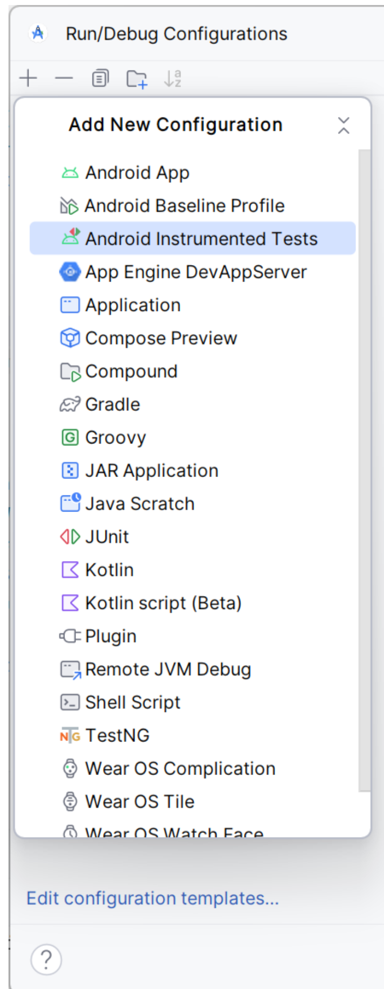
- Unit tests
  - directory: src/test/java
  - library: JUnit https://junit.org/junit4/ (ver. 4) or https://junit.org/junit5/ (ver. 5)
  - doc: https://developer.android.com/training/testing/local-tests
- UI tests (instrumented tests)
  - directory: src/androidTest/java
  - library: Jetpack Compose https://developer.android.com/develop/ui/compose/testing/testing-cheatsheet
  - doc: https://developer.android.com/training/testing/instrumented-tests
- Tutorials
  - https://developer.android.com/codelabs/basic-android-kotlin-compose-test-viewmodel
  - https://developer.android.com/codelabs/basic-android-kotlin-compose-write-automated-tests

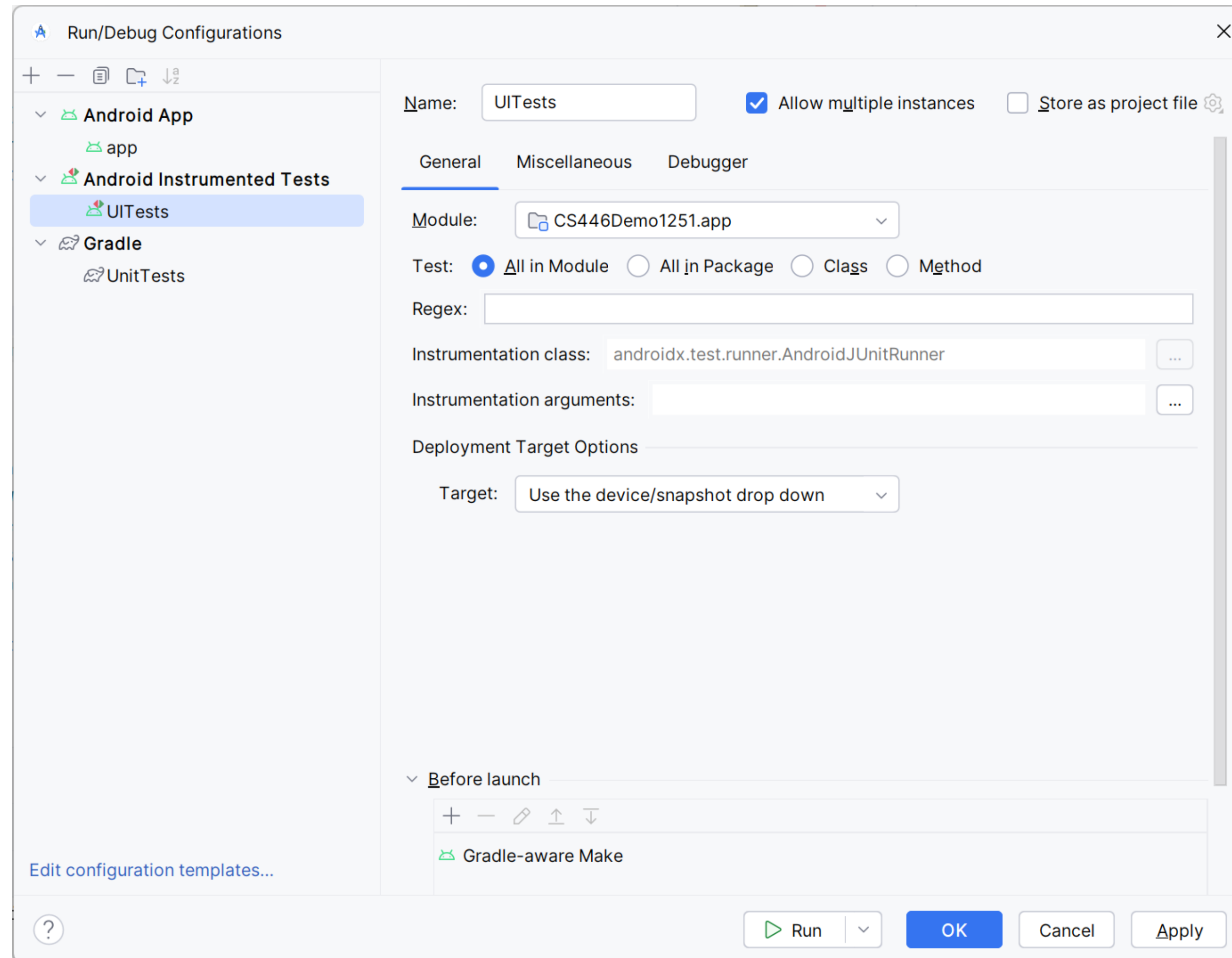
Demo – setup unit tests run in Android Studio





Demo – setup UI tests run in Android Studio





Demo – run tests with coverage

Pixel 7 Pro API 34 | UnitTests

Run 'UnitTests' with Coverage

Configuration

Edit...

Delete

```
package ca.uwaterloo.cs446.tiptime

import kotlin.math.ceil

class TipCalculator(
    var amount: Double = 0.0,
    var tipPercent: Double = 0.0,
    var roundUp: Boolean = false,
) {
    fun calculateTip(): Double {
        var tip = tipPercent / 100 * amount
        if (roundUp) {
            tip = ceil(tip)
        }
        return tip
    }
}
```

UnitTests

Coverage | UnitTests

Element	Class, %	Method, %	Line, %	Branch, %
ca.uwaterloo.cs446	37% (3/8)	33% (7/21)	26% (30/114)	4% (5/120)
tiptime	42% (3/7)	36% (7/19)	27% (30/110)	4% (5/120)
TipCalculator	100% (1/1)	100% (2/2)	85% (6/7)	50% (1/2)
TipTimeScreenKt	0% (0/4)	0% (0/11)	0% (0/75)	0% (0/106)
TipTimeUiState	100% (1/1)	100% (1/1)	100% (4/4)	100% (0/0)
TipTimeViewModel	100% (1/1)	80% (4/5)	83% (20/24)	33% (4/12)
MainActivity	0% (0/1)	0% (0/2)	0% (0/4)	100% (0/0)

Agenda (recap)

- Testing basics
- Categories
  - by granularity: unit, integration, e2e
  - by subject: functional, UI, performance
  - by methodology: regression, random, differential, metamorphic
- Testing libraries and techniques (w/ Demo)
  - running
  - measuring coverage