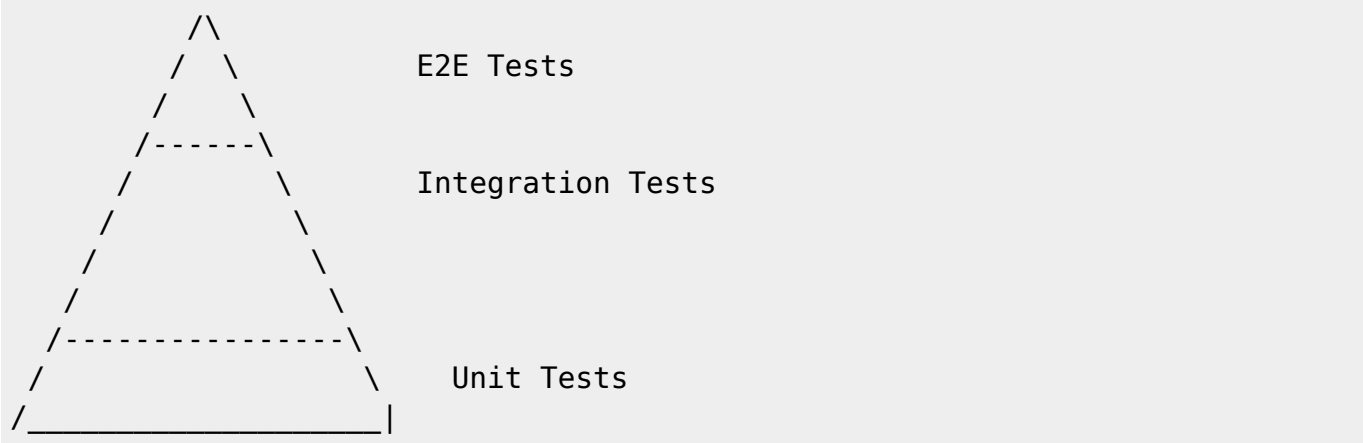
Estrategia de Testing en PAE

Pirámide de testing



Distribución recomendada

- **Unit Tests**: 70% (rápidos, aislados)
- **Integration Tests**: 20% (módulos, servicios)
- **E2E Tests**: 10% (flujos completos)

Unit Tests

Estructura

```
Machine/src/
├── test/
│   └── java/co/ada/paemachine/
│       ├── viewmodels/
│       │   └── LaboratoryViewModelTest.kt
│       ├── state/
│       │   └── StateManagerTest.kt
│       └── services/
│           └── DeliveryServiceTest.kt
```

Ejemplo: StateManager

```kotlin
// MachineDomain/src/test/kotlin/co/ada/domain/state/StateManagerTest.kt

@RunWith(RobolectricTestRunner::class)
class StateManagerTest {
    private lateinit var manager: StateManager
```

```
    private val mockRepository: Repository = mock()
    private val mockHardware: Hardware = mock()

    @Before
    fun setup() {
        manager = StateManager(mockRepository, mockHardware)
    }

    @Test
    fun testStateTransitionWaitingForWeight() {
        // Arrange
        val context = RuntimeEnvironment.getApplication()

        // Act
        val initialized = manager.init(context)
        val currentState = manager.getCurrentState()

        // Assert
        assertTrue(initialized)
        assertEquals("WaitingForWeight", currentState)
    }

    @Test
    fun testStateTransitionOnSuccess() {
        // Arrange
        val results = mutableListOf<String>()
        manager.init(RuntimeEnvironment.getApplication())

        // Act
        manager.cycle()
        results.add(manager.getCurrentState())

        // Assert
        assertTrue(results.isNotEmpty())
    }

    @Test
    fun testStateRetryOnError() {
        // Arrange
        val manager = StateManager(mockRepository, mockHardware)

        // Act
        manager.init(RuntimeEnvironment.getApplication())

        // TODO: simular error y verificar retry

        // Assert
    }
}
```

Ejemplo: BeneficiaryService

```
// MachineData/src/test/kotlin/co/ada/data/services/BeneficiaryServiceTest.kt

class BeneficiaryServiceTest {
    private val mockRepository: Repository = mock()
    private lateinit var service: BeneficiaryService

    @Before
    fun setup() {
        service = BeneficiaryService(mockRepository)
    }

    @Test
    fun testFindSimilar() {
        // Arrange
        val embedding = FloatArray(512) { Random.nextFloat() }
        val beneficiaries = listOf(
            Beneficiary(id = 1, name = "Juan", embedding = embedding),
            Beneficiary(id = 2, name = "María", embedding = embedding)
        )

        `when`(mockRepository.beneficiaries.nearestNeighbors(embedding, 5))
            .thenReturn(beneficiaries)

        // Act
        val results = service.findSimilar(embedding)

        // Assert
        assertEquals(2, results.size)
        verify(mockRepository.beneficiaries).nearestNeighbors(embedding, 5)
    }

    @Test
    fun testCreateBeneficiary() {
        // Arrange
        val beneficiary = Beneficiary(
            id = 1,
            name = "Juan",
            embedding = FloatArray(512)
        )

        // Act
        service.create(beneficiary)

        // Assert
        verify(mockRepository.beneficiaries).create(beneficiary)
    }
```

```
}
```

Test fixtures

```
// shared/src/test/kotlin/co/ada/test/fixtures/

object BeneficiaryFixtures {
    fun makeBeneficiary(
        id: Long = 1,
        name: String = "Juan",
        embedding: FloatArray = FloatArray(512)
    ) = Beneficiary(
        id = id,
        name = name,
        embedding = embedding
    )
}

object DeliveryFixtures {
    fun makeDelivery(
        id: Long = 1,
        weight: Float = 10.5f,
        similarity: Float = 0.95f
    ) = Delivery(
        id = id,
        weight = weight,
        similarity = similarity
    )
}
```

Android Tests (Instrumented)

Estructura

```
Machine/src/
├── androidTest/
│   └── java/co/ada/paemachine/
│       ├── screens/
│       │   └── LaboratoryScreenTest.kt
│       └── integration/
│           └── DeliveryFlowTest.kt
```

Ejemplo: Composable UI Test

```
//
```

Machine/src/androidTest/kotlin/co/ada/paemachine/screens/LaboratoryScreenTest.kt

```kotlin
@RunWith(AndroidJUnit4::class)
class LaboratoryScreenTest {
    @get:Rule
    val composeTestRule = createComposeRule()

    @Test
    fun testCapturButtonVisibile() {
        // Arrange
        composeTestRule.setContent {
            LaboratoryScreen(viewModel = mockViewModel)
        }

        // Act & Assert
        composeTestRule.onNodeWithText("Capturar").assertIsDisplayed()
    }

    @Test
    fun testCapturButtonClick() {
        // Arrange
        composeTestRule.setContent {
            LaboratoryScreen(viewModel = mockViewModel)
        }

        // Act
        composeTestRule.onNodeWithText("Capturar").performClick()

        // Assert
        verify(mockViewModel).startCapture()
    }

    @Test
    fun testStateProgression() {
        // Arrange
        composeTestRule.setContent {
            LaboratoryScreen(viewModel = mockViewModel)
        }

        // Act
        composeTestRule.onNodeWithText("Capturar").performClick()
        composeTestRule.waitUntil(5000) {
            // esperar estado cambio
            true
        }

        // Assert
        composeTestRule.onNodeWithText("Capturando
rostro").assertIsDisplayed()
    }
```

```
}
```

Database Test

```
// MachineData/src/androidTest/kotlin/co/ada/data/DeliveryDatabaseTest.kt

@RunWith(AndroidJUnit4::class)
class DeliveryDatabaseTest {
    private lateinit var db: DeliveryDatabase
    private lateinit var dao: DeliveryDao

    @Before
    fun createDb() {
        val context =
InstrumentationRegistry.getInstrumentation().targetContext
        db = Room.inMemoryDatabaseBuilder(context,
DeliveryDatabase::class.java)
            .allowMainThreadQueries()
            .build()
        dao = db.deliveryDao()
    }

    @After
    fun closeDb() {
        db.close()
    }

    @Test
    fun testInsertAndGetDelivery() = runBlocking {
        // Arrange
        val delivery = Delivery(
            id = 1,
            weight = 10.5f,
            similarity = 0.95f
        )

        // Act
        dao.insert(delivery)
        val retrieved = dao.getById(1)

        // Assert
        assertNotNull(retrieved)
        assertEquals(10.5f, retrieved?.weight)
    }
}
```

Integration Tests

Ejemplo: DeliverySyncWorker

```
// RutaPAE/src/androidTest/kotlin/co/ada/rutapae/workers/DeliverySyncWorkerTest.kt

@RunWith(AndroidJUnit4::class)
class DeliverySyncWorkerTest {
    @get:Rule
    val instantTaskExecutorRule = InstantTaskExecutorRule()

    private lateinit var context: Context
    private lateinit var testDriver: WorkManagerTestInitHelper

    @Before
    fun setup() {
        context = InstrumentationRegistry.getInstrumentation().targetContext
        testDriver = WorkManagerTestInitHelper(context)
    }

    @Test
    fun testSyncWorkerSuccess() {
        // Arrange
        val machineId = 1L
        val request = OneTimeWorkRequestBuilder<DeliverySyncWorker>()
            .setInputData(
                workDataOf("machineId" to machineId)
            )
            .build()

        // Act
        WorkManager.getInstance(context).enqueueUniqueWork(
            "sync_$machineId",
            ExistingWorkPolicy.KEEP,
            request
        )

        testDriver.periodicallyEnqueue(request.id)
        testDriver.work()

        // Assert
        testDriver.getWorkInfoManager().getWorkInfoById(request.id).get()
            .also { info ->
                assertEquals(WorkInfo.State.SUCCEEDED, info.state)
            }
    }
```

```
}
```

E2E Tests (Espresso)

Ejemplo: Flujo completo

```
// Machine/src/androidTest/kotlin/co/ada/paemachine/DeliveryFlowTest.kt

@RunWith(AndroidJUnit4::class)
class DeliveryFlowTest {
    @get:Rule
    val activityRule = ActivityScenarioRule(MachineActivity::class.java)

    @Test
    fun testCompleteDeliveryFlow() {
        // 1. Navega a Laboratory Screen
        onView(withId(R.id.laboratory_screen)).check(matches(isDisplayed()))

        // 2. Presiona Capturar
        onView(withText("Capturar")).perform(click())

        // 3. Espera a WaitingForWeight
        onView(withText("Esperando peso..."))
            .check(matches(isDisplayed()))

        // 4. Simula input de balanza
        // (requiere mock de Hardware)

        // 5. Verifica transición a CaptureImages
        onView(withText("Capturando imagen..."))
            .check(matches(isDisplayed()))

        // 6. Verifica progreso completo
        onView(withText("Entrega guardada"))
            .check(matches(isDisplayed()))
    }
}
```

Mocking y Testing Doubles

Mockito

```
// Crear mock
val mockRepository: Repository = mock()
```

```
// Configurar comportamiento
`when`(mockRepository.getDelivery(1)).thenReturn(DeliveryFixtures.makeDelive
ry())

// Verificar llamadas
verify(mockRepository, times(1)).getDelivery(1)
verify(mockRepository, never()).deleteDelivery(anyLong())
```

Manual Test Doubles

```
class FakeRepository : Repository {
    private val deliveries = mutableMapOf<Long, Delivery>()

    override fun getDelivery(id: Long): Delivery? = deliveries[id]
    override fun create(delivery: Delivery): Delivery {
        deliveries[delivery.id] = delivery
        return delivery
    }
    override fun delete(id: Long): Boolean {
        return deliveries.remove(id) != null
    }
}

class StubHardware : Hardware {
    override fun getWeight(): Float = 10.5f
    override fun ledOn() { /* no-op */ }
}
```

Coverage de código

Ejecutar coverage

```
./gradlew jacocoTestReport

# Report en build/reports/jacoco/
```

Configurar jacoco

```
// build.gradle.kts
plugins {
    id 'jacoco'
}

jacoco {
    toolVersion = "0.8.8"
}
```

```
task jacocoTestReport(type: JacocoReport) {
    dependsOn test
    reports {
        xml.enabled true
        html.enabled true
    }
}
```

Mínimo coverage requerido

- **Domain**: 80%+
- **Data**: 70%+
- **UI**: 50%

Continuous Integration Testing

GitHub Actions

```
name: Tests

on: [push, pull_request]

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v3
    - name: Set up JDK 17
      uses: actions/setup-java@v3
      with:
        java-version: '17'
    - name: Run unit tests
      run: ./gradlew test
    - name: Generate coverage
      run: ./gradlew jacocoTestReport
    - name: Upload coverage
      uses: codecov/codecov-action@v3
      with:
        files: ./build/reports/jacoco/
```

Performance Testing

Memory profiler

```
# Conectar dispositivo
adb shell am start-profiler [process] [file]

# Capturar dump
adb shell am dump-heap [process] [file]
```

CPU profiler

```
# Android Studio: Profiler → CPU
# o vía Logcat: adb logcat | grep "Trace"
```

Debugging de tests

Breakpoints en tests

- Android Studio: Click en número de línea
- Run → Debug 'TestName'
- F9 para continuar

Logs en tests

```
@Test
fun myTest() {
    android.util.Log.d("TEST", "Starting test")
    // ...
}
```

Ver logs:

```
adb logcat | grep TEST
```

Best practices de testing

1. **AAA Pattern**: Arrange, Act, Assert
2. **One assertion per test**: facilita debugging
3. **Descriptive names**: `testWeightInputUpdatesUI` not `test1`
4. **Isolate dependencies**: usar mocks
5. **Mock external systems**: HTTP, BD, hardware
6. **Fast tests**: < 100ms para unit tests
7. **Deterministic**: mismo resultado siempre
8. **Independientes**: no dependen del orden
9. **Limpieza**: reset mocks y estado en @After

Test Data Builders

```
class DeliveryBuilder {
    private var id: Long = 1
    private var weight: Float = 10.5f
    private var similarity: Float = 0.95f

    fun withId(id: Long) = apply { this.id = id }
    fun withWeight(weight: Float) = apply { this.weight = weight }
    fun withSimilarity(similarity: Float) = apply { this.similarity =
similarity }

    fun build() = Delivery(
        id = id,
        weight = weight,
        similarity = similarity
    )
}

// Uso
val delivery = DeliveryBuilder()
    .withId(2)
    .withWeight(15f)
    .build()
```

Parametrized Tests

```
@RunWith(Parameterized::class)
class WeightValidationTest(
    val input: Float,
    val expected: Boolean
) {
    companion object {
        @Parameterized.Parameters
        @JvmStatic
        fun data() = listOf(
            arrayOf(0f, false),    // peso 0 es inválido
            arrayOf(5.5f, true),   // peso válido
            arrayOf(100f, true),   // peso válido
            arrayOf(-1f, false)    // peso negativo inválido
        )
    }

    @Test
    fun testWeightValidation() {
        val result = WeightValidator.isValid(input)
        assertEquals(expected, result)
```

```
    }
}
```