Testing Efectivo en Java

Círculo Siete Capacitación

Clase 4 de 12

12 Agosto 2025

Acceptance Test-Driven Development (ATDD)

- Define las pruebas de aceptación antes de escribir código.
- Involucra a negocio, QA y desarrollo para acordar qué significa que una funcionalidad esté “terminada”.
- Utiliza un lenguaje comprensible y estructurado para que todos entiendan los requisitos.
- Idea clave: ATDD asegura que construimos lo que el cliente realmente necesita, no lo que el desarrollador asumió.

En breve ATDD

- Negocio describe la necesidad en un lenguaje comprensible (generalmente Gherkin).
- QA transforma estas necesidades en pruebas de aceptación.
- Desarrollo implementa el código que haga pasar esas pruebas.
- ATDD se diferencia de TDD en que TDD parte de pruebas unitarias para guiar el diseño técnico, mientras que ATDD parte de pruebas de aceptación para guiar la funcionalidad completa.

Ciclo de ATDD

- Definir pruebas de aceptación
  - Se redactan los criterios de aceptación en lenguaje natural estructurado (ej. Gherkin).
- Automatizar las pruebas
  - Antes de escribir código, se implementan los step definitions que inicialmente fallarán.
- Desarrollar la funcionalidad
  - Escribir el código mínimo necesario para que las pruebas pasen.
- Refactorizar
  - Mejorar el código manteniendo las pruebas verdes.
- Este ciclo sigue el mismo espíritu Red → Green → Refactor de TDD, pero a nivel de comportamiento del sistema.

Beneficios clave

- Comunicación clara: todos entienden lo que significa "hecho".
- Prevención de malentendidos: reduce el riesgo de entregar algo que no es lo esperado.
- Pruebas vivas: los criterios de aceptación son ejecutables y siempre actualizados.
- Facilita BDD: ATDD es un puente natural hacia el Behavior-Driven Development.

Herramientas comunes en Java

- Cucumber → Automatiza escenarios escritos en Gherkin.
- Serenity BDD → Integra ATDD/BDD con reportes detallados.
- JUnit 5 → Base de ejecución de las pruebas.
- Selenium / RestAssured → Interacción con UI o APIs.

Comparativa TDD y ATDD

Característica	TDD	ATDD
Enfoque inicial	Pruebas unitarias	Pruebas de aceptación
Nivel de prueba	Bajo (código)	Alto (funcionalidad)
Participantes	Desarrollo	Desarrollo, QA, Negocio
Lenguaje	Código	Natural Estructurado

¿Qué es Behavior-Driven Development (BDD)?

- Behavior-Driven Development (BDD) es una práctica de desarrollo de software que extiende Test-Driven Development (TDD), enfocándose en la colaboración entre negocio, desarrollo y QA para definir el comportamiento esperado de un sistema en un lenguaje entendible por todos.
- Su objetivo principal es alinear el desarrollo del software con las necesidades del negocio mediante ejemplos concretos que describen qué debe hacer el sistema, no cómo implementarlo.

Principios clave de BDD

- Lenguaje ubicuo y compartido
  - Todos los involucrados (desarrolladores, testers, analistas, negocio) usan un mismo lenguaje claro para describir comportamientos.
- Especificaciones como documentación viva
  - Las pruebas en BDD funcionan como especificaciones siempre actualizadas.
- Colaboración temprana
  - Antes de escribir código, se definen los comportamientos esperados de forma conjunta.
- Ejemplos concretos
  - Se describen casos reales que ilustran reglas de negocio y escenarios.

Relación con TDD

- En TDD, las pruebas se centran en unidades de código y son escritas en un lenguaje técnico.
- En BDD, las pruebas se enfocan en el comportamiento del sistema desde el punto de vista del usuario, y suelen expresarse en un lenguaje casi natural.
- BDD no reemplaza a TDD, sino que lo complementa agregando una capa de comunicación y claridad.

Formato Gherkin

- En BDD, es común usar el lenguaje Gherkin para describir escenarios en un formato estándar:

```
Feature: Cálculo de envío

  Scenario: Cliente obtiene envío gratis
    Given que el cliente tiene un carrito con valor de 600 pesos
    When procede al pago
    Then el costo de envío debe ser 0
```

Gherkin

- Gherkin es un lenguaje de especificación legible por humanos, usado para describir el comportamiento de una aplicación de forma clara y entendible tanto para desarrolladores como para personas no técnicas.
- Su propósito es conectar especificaciones con pruebas automatizadas siguiendo un enfoque BDD (Behavior-Driven Development).
- En el ecosistema Java, Gherkin se utiliza principalmente junto con Cucumber u otras herramientas BDD, permitiendo escribir escenarios que luego se enlazan a código Java que ejecuta las pruebas.

Ventajas de Gherkin

- Comprensible por personas no técnicas.
- Facilita la automatización con herramientas como Cucumber, SpecFlow o Behave.
- Sirve como contrato entre las partes.

Proceso típico en BDD

- **Conversación**: Negocio, QA y desarrollo discuten el comportamiento deseado.
- **Especificación**: Se documentan los escenarios usando un lenguaje claro (ej. Gherkin).
- **Automatización**: Se implementan pruebas automatizadas que validan esos escenarios.
- **Desarrollo**: Se escribe el código que hace pasar las pruebas.
- **Refactorización**: Se mejora el código manteniendo las pruebas verdes.

Beneficios

- Reduce malentendidos entre negocio y desarrollo.
- Crea documentación viva del sistema.
- Fomenta el desarrollo guiado por reglas de negocio.
- Aumenta la confianza al hacer cambios.

Cucumber

- Aslak Hellesøy su creador, buscaba un nombre simpático y memorable, que no sonara intimidante y que no tuviera connotaciones técnicas.
- En sus propias palabras, “me gustan los pepinos, y Cucumber suena divertido”. No hay un acrónimo oculto ni una referencia a patrones de diseño — fue un nombre juguetón para una herramienta que buscaba hacer las pruebas más amigables.
- De hecho, la idea se alineaba con el espíritu de Cucumber:
  - Hacer que las pruebas fueran más “frescas” y fáciles de digerir para todos, incluso personas no técnicas.
  - Romper con la rigidez de los nombres tradicionales de frameworks.

Gherkin

- En inglés, gherkin significa "pepinillo", esos pepinos pequeños encurtidos que se usan en hamburguesas o ensaladas. La relación es directa:
  - Cucumber = pepino fresco.
  - Gherkin = pepino encurtido.
- La broma interna era que Gherkin es el “lenguaje” que alimenta a Cucumber, como si estuvieras “preparando” el pepino antes de servirlo.
- Gherkin es un lenguaje estructurado pero legible para humanos que describe casos de prueba en formato Given–When–Then.
- El nombre refuerza la idea de que todo el ecosistema de Cucumber se mantiene “temáticamente” alrededor de los pepinos.
- Aslak Hellesøy contó que le gustaba que la comunidad se divirtiera con la metáfora culinaria, y hasta en las primeras versiones había bromas en la documentación sobre “*pickling your features*”.

Guía práctica:
BDD en Java con Cucumber

- 1. Visión general
  - En Java, BDD se suele implementar con Cucumber como herramienta principal.
  - Gherkin → Define escenarios legibles por negocio.
  - Step Definitions (Java) → Conectan escenarios con código ejecutable.
  - JUnit/TestNG → Orquestan la ejecución de las pruebas.

2. Estructura típica del proyecto

- Con Maven o Gradle, un proyecto BDD suele verse así:

```
src
 ├── main
 │    └── java
 │         └── com.miempresa.app
 │              └── ... (código de producción)
 └── test
      ├── java
      │    └── com.miempresa.app.steps    # Step Definitions
      │    └── com.miempresa.app.runners  # Clases de ejecución
      └── resources
           └── features                   # Archivos .feature con Gherkin
```

3. Código a escribir

- Archivos de features
- Runner
- Steps
- Finalmente el código de implementación

Ejercicio

- Laboratorio de ATDD
  - Servicio de cálculo de costo de envío
  - Carpeta ***labs***

Buenas prácticas en BDD con Java

- Usar lenguaje de negocio en los .feature.
- Mantener steps pequeños y reutilizables.
- Evitar lógica compleja en los steps → delegar a clases de dominio.
- Organizar features por módulo o funcionalidad.
- Integrar Cucumber en el pipeline de CI/CD para validaciones automáticas.