¿Qué es BDD?

BDD refiere a *Behavior Driven Development*, o sea, desarrollo dirigido por comportamiento. Como bien lo indica su nombre, no se trata de una técnica de testing, sino que es una estrategia de desarrollo (así como TDD, que es *test driven development*). Lo que plantea es definir un lenguaje común para el negocio y para los técnicos, y utilizar eso como parte inicial del desarrollo y el testing. Por esto es que es importante saber como parte del team entiendas bien qué es BDD.

BDD y TDD

BDD es una evolución del TDD. En TDD se enfoca a la prueba unitaria, en cambio en BDD se enfoca en la prueba de más alto nivel, la prueba funcional, la de aceptación, el foco está en cumplir con el negocio y no solo con el código.
Ambos enfoques podrían convivir, ya que se podría especificar una prueba de aceptación, y luego refinar en pruebas unitarias al momento de codificar esa funcionalidad en las distintas capas. Tal vez una ventaja clara de BDD es que los desarrolladores se enfocan en ver qué es lo que tiene que funcionar y de qué forma a nivel de negocio.

BDD en metodologías ágiles

Esta estrategia encaja bien en las metodologías ágiles, ya que generalmente en ellas se especifican los requerimientos como historias de usuario. Estas historias de usuario deberán tener sus criterios de aceptación, y de ahí se desprenden pruebas de aceptación, las cuales pueden ser escritas directamente en lenguaje Gherkin (leer lo que sigue).

Lenguaje común para negocio y técnicos: Gherkin

Gherkin es un lenguaje común, que lo puede escribir alguien sin conocimientos en programación, pero que lo puede comprender también un programa, de forma tal de utilizarlo como especificación de pruebas. Típicamente, estas pruebas se van a guardar en archivos “.feature”, los cuales deberían estar versionados junto al código fuente del sistema que se está probando. Este es un ejemplo simple tomado de Cucumber:

```
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

  Scenario: A different situation
      ...
```

En estos archivos se especifica:

- **Feature**: nombre de la funcionalidad que vamos a probar, el título de la prueba.
- **Scenario**: habrá uno por cada prueba que se quiera especificar para esta funcionalidad.
- **Given**: acá se marca el contexto, las precondiciones.
- **When**: se especifican las acciones que se van a ejecutar.
- **Then**: y acá se especifica el resultado esperado, las validaciones a realizar.

Puede haber un *feature* por archivo y este contendrá distintos escenarios de prueba.
Nota: Gherkin soporta muchos lenguajes, así que si bien lo más común es verlo en inglés, se podría hacer en español también.

¿Quién escribe las pruebas de aceptación?

Esto creo que depende mucho de cada equipo, pero podrá ser desde el Product Owner o alguien del equipo, ya sea tester o desarrollador. Tal vez una buena opción es hacerlo en conjunto, al menos el brainstorming, como parte del refinamiento de las historias. Luego, el tester se encargaría de asegurarse que estén bien escritas, completas, con el cubrimiento de acuerdo a la estrategia de pruebas definidas.

BDD y testeabiliidad

El enfoque BDD favorece la testeabilidad del sistema. Me atrevo a decir que la mejor forma de hacer funcionar este esquema es con una arquitectura del estilo de microservicios, que permita agregar y trabajar

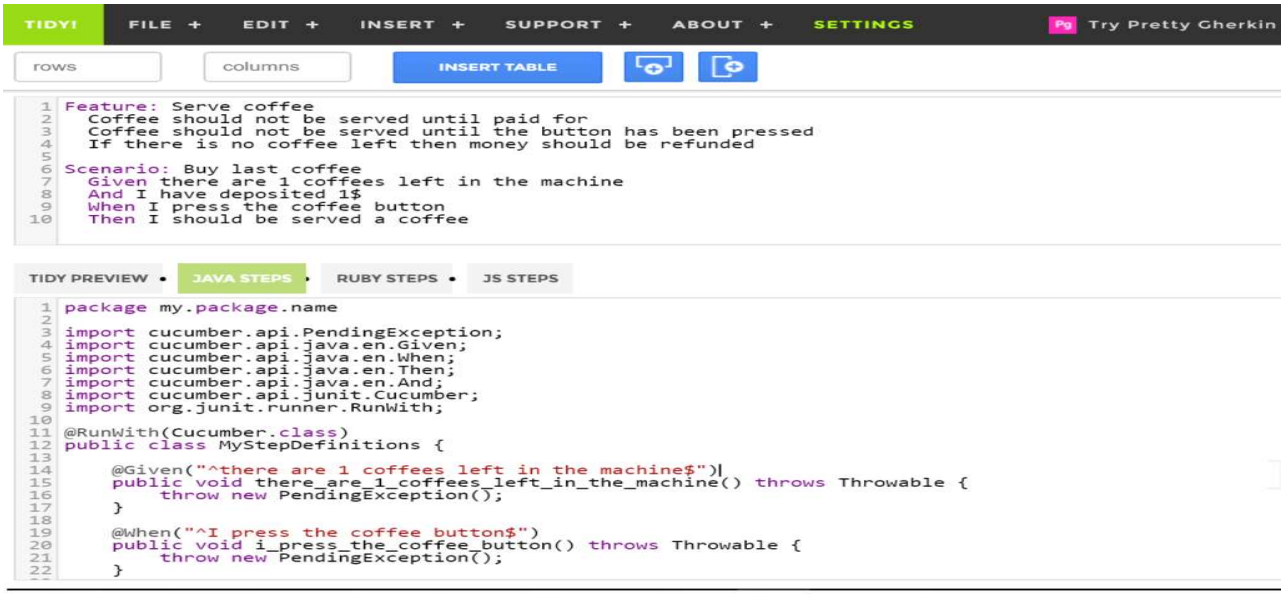
una funcionalidad por todas sus capas en forma independiente. Este esquema facilita las pruebas, entonces mejora la testeabilidad.

No debería haber una historia que sea “implementar la funcionalidad XX en la capa de acceso a datos”. Eso no está orientado al negocio. BDD funciona mejor en un sistema testeable, donde la torta está bien cortada, enfocando las historias a una funcionalidad del negocio, y al implementarla (para poder darla por terminada) se deben completar todas sus capas.

BDD para automatizar pruebas

El lenguaje Gherkin es simplemente texto con algunas palabras clave y algo de estructura, que luego hay herramientas como Cucumber , que permiten implementar una capa de conexión entre esa especificación de prueba y la interfaz del sistema que se quiere probar, pudiendo así utilizar eso como los pasos de una prueba automatizada.

Para comenzar a practicar un poco, hay un plugin de Google Chrome muy bueno llamado *Tidy Gherkin*. Uno puede escribir las features en el campo de texto de arriba, y muestra cómo es el código necesario para ejecutar esos pasos en distintos lenguajes (Java, Ruby y JavaScript).



Conclusión

Una buena combinación para trabajar BDD podría ser:

- Tener una arquitectura que permita trabajar de forma modular, y que estas historias se completen cuando se completa de punta a punta.

- Cuando se refinan las historias, escribir las pruebas de aceptación en Gherkin (puede hacerlo un tester/analista/QA sin conocimiento en programación).
- Se trabaja durante el sprint en el desarrollo. Al inicio del sprint, mientras no hay ninguna historia pronta para probar, el tester/analista/QA que automatiza puede trabajar en quitar deuda técnica de testing.
- Una vez que una historia está lista para probar, se realiza test exploratorio, y si se considera que se puede automatizar, se implementa la capa que conecta el archivo feature con la interfaz del sistema a probar. Esto es usando Cucumber con algún lenguaje (Java, Ruby, etc.), con algún driver apropiado, que puede ser Selenium si probaremos una web, Appium si es algo mobile, SoapUI para servicios web.
- Versionar el código de las pruebas, el código del sistema y las features con el mismo criterio.