

TDD

Principios básicos del TDD



■ Contenido

Conceptos que vamos a cubrir

- Que es el TDD
- Otras aproximaciones al TDD
 - BDD
 - ATDD



Qué es TDD



■ TDD - Qué es

TDD === “Test Driven Development”

- Es una forma de desarrollo iterativa que consiste en la realización de los tests en primer lugar, desarrollo de la funcionalidad y refactorizar
- Una descripción más sencilla es crear el test y que falle y desarrollar la funcionalidad hasta que el test esté en verde, posteriormente si cambia esa funcionalidad habría que adaptar primero el test y que falle hasta que realicemos la refactorización



■ TDD - Las leyes

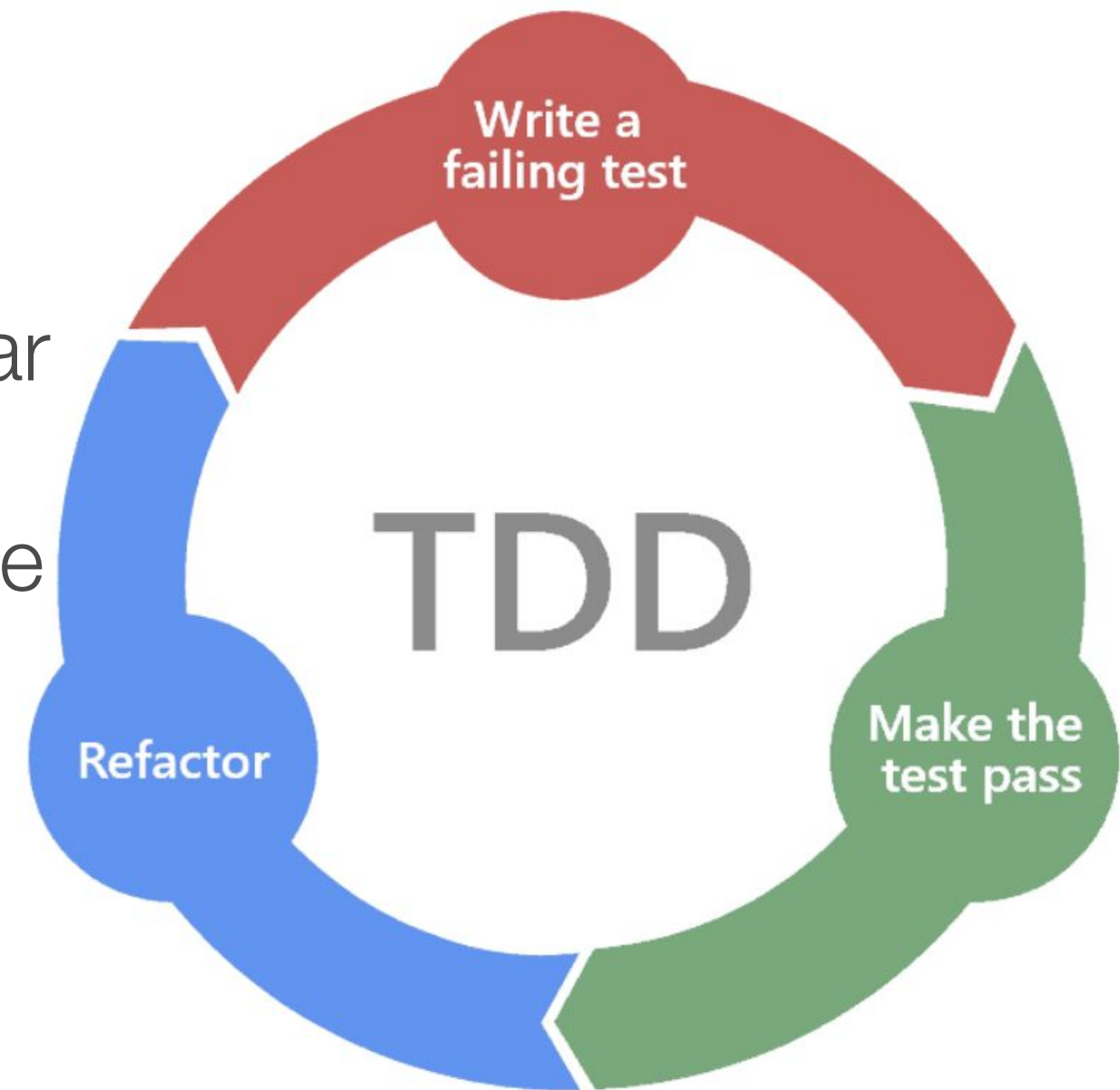
Robert C. Martin describe la esencia del TDD como un proceso que atiende a las siguientes tres reglas:

- No escribirás código de producción sin antes escribir un test que falle.
- No escribirás más de un test unitario suficiente para fallar (y no compilar es fallar).
- No escribirás más código del necesario para hacer pasar el test.



■ TDD - Ciclo Red-Green-Refactor

1. Red: Escribimos un test que falle
 - a. Puede ser unit o integración
2. Green: Implementamos lo necesario
3. Refactor: Analizar si podemos mejorar el código que hemos hecho
4. Cerrado el ciclo, pasamos al siguiente requisito



■ TDD - Reglas (I)

1. TDD especifica una serie de reglas que han de cumplirse:
2. Tener bien definidos los requisitos de la funcionalidad a realizar
 - a. Un mala definición de los requisitos provocaría que no se siguiera TDD de forma adecuada e implicaría una pérdida innecesaria de tiempo
3. Contemplar todos los casos posibles, tanto de éxito como de error en los criterios de aceptación de la funcionalidad



■ TDD - Reglas (II)

4. Cómo vamos a diseñar el test
 - a. Para realizar un buen test debemos ceñirnos a testear únicamente la lógica de cada elemento, utilizando mocks para abstraernos de otras posibles capas o servicios que necesitemos utilizar. Esto es flexible, por ello es el desarrollador el encargado de decidir si es mejor utilizar mocks o usar los servicios reales en el test
5. Qué queremos probar
 - a. Esto implica también el punto de vista del desarrollador, ya que cada uno puede pensar que se debería testear sólo una cosa en especial y otro puede pensar que deben testearse más



■ TDD - Reglas (III)

6. ¿Cuántos test son necesarios?
 - a. El número de test nunca está especificado, se basa en el número de casuísticas que contemple nuestra funcionalidad



■ TDD - Principios

Algunos de los principios en los que se basa TDD son los conocidos como principios SOLID:

- (S) Principio de responsabilidad simple (Single Responsibility Principle)
 - Una clase o módulo debe tener una única responsabilidad
- (O) Principio de abierto/cerrado (OCP)
 - Una clase debe permitir ser extendida sin necesidad de ser modificada
- (L) Principio de sustitución de Liskov (LSP)
 - Si una función recibe un parámetro de un tipo, esta función debe ejecutarse correctamente si recibe un parámetro de ese tipo de alguna de sus subclases
- (I) Principio de segregación de interfaces (ISP)
 - No debemos obligar a clases o interfaces a depender de otras que no necesitan
- (D) Principio de inversión de dependencia (DIP)
 - Son técnicas para lidiar con las colaboraciones entre clases produciendo código limpio y reutilizable. El módulo A no debe depender del módulo B para su correcto funcionamiento



■ TDD - Beneficios

- Mayor calidad en el código desarrollado
- Diseño orientado a las necesidades
- Simplicidad
- Menos redundancia
- Mayor productividad al necesitar menor tiempo de debugging
- Se reduce el número de errores

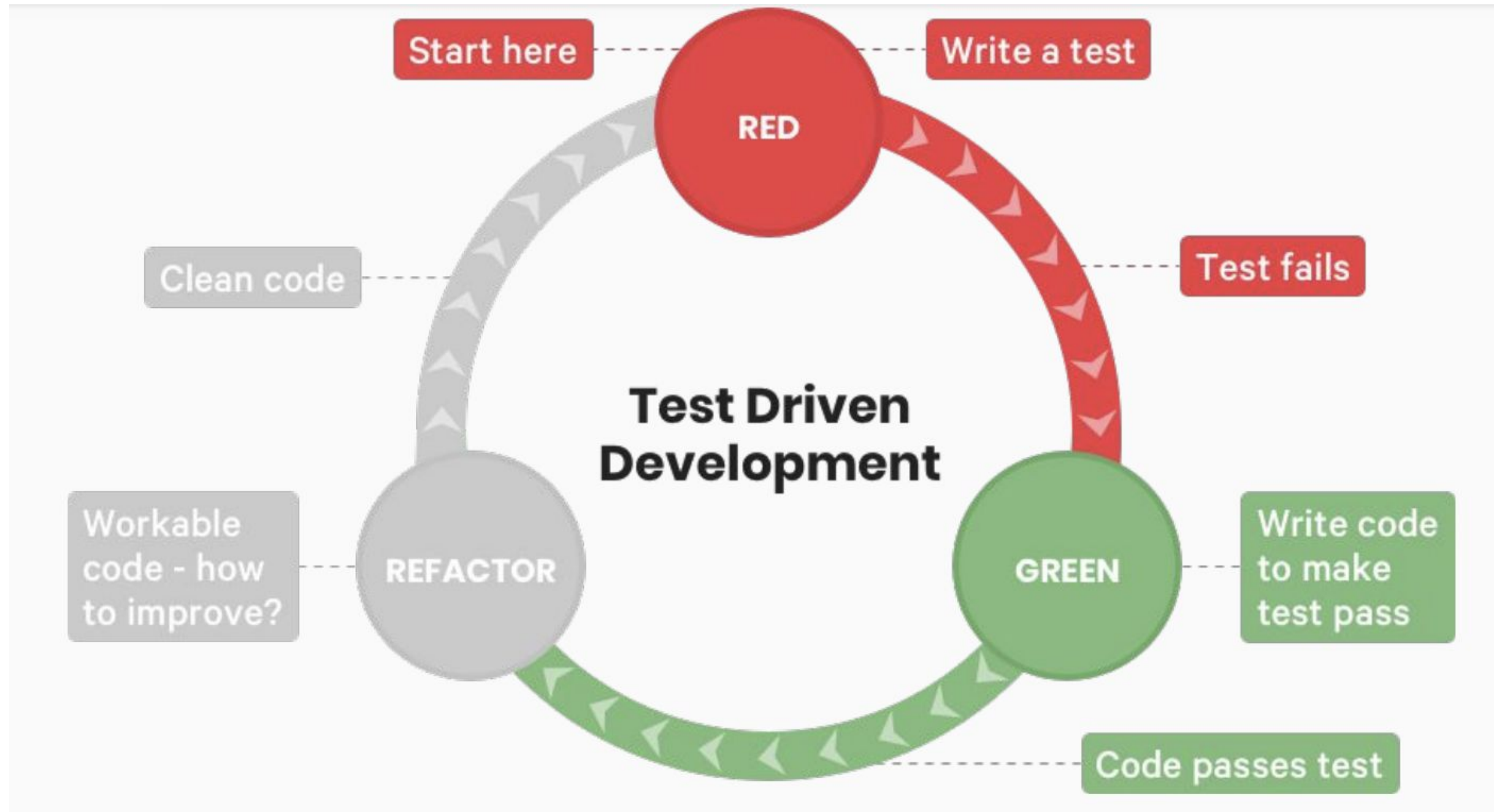


■ TDD - Contrapartidas

- Gran curva de aprendizaje
- Se puede perder la visión general del proyecto
- Errores no identificados en el alcance de la funcionalidad
- Si se usa mal, podemos afectar servicios como bases de datos
- Es difícil de implementar en el front-end, puesto que está diseñado para el testeo de la lógica de negocio
- Gran inversión de tiempo



■ TDD - resumen



Otras aproximaciones al TDD



■ BDD

BDD === “Behavior Driven Development”

- Es un proceso de desarrollo de software basado en el testing
- BDD busca un lenguaje común para unir la parte técnica y la de negocio
- En BDD las pruebas de aceptación son las conocidas en agile como historias de usuario
- El objetivo de BDD es un lenguaje específico que permite describir un comportamiento en tu app sin importar cómo ese comportamiento está implementado
- Usa Gherkins para describir los test y que sean lo más descriptible posible



■ BDD - Herramientas

El framework más popular es Cucumber: <https://cucumber.io/>

- Existe tanto para java, como javascript, Ruby y Kotlin

La sintaxis Gherkins es la siguiente

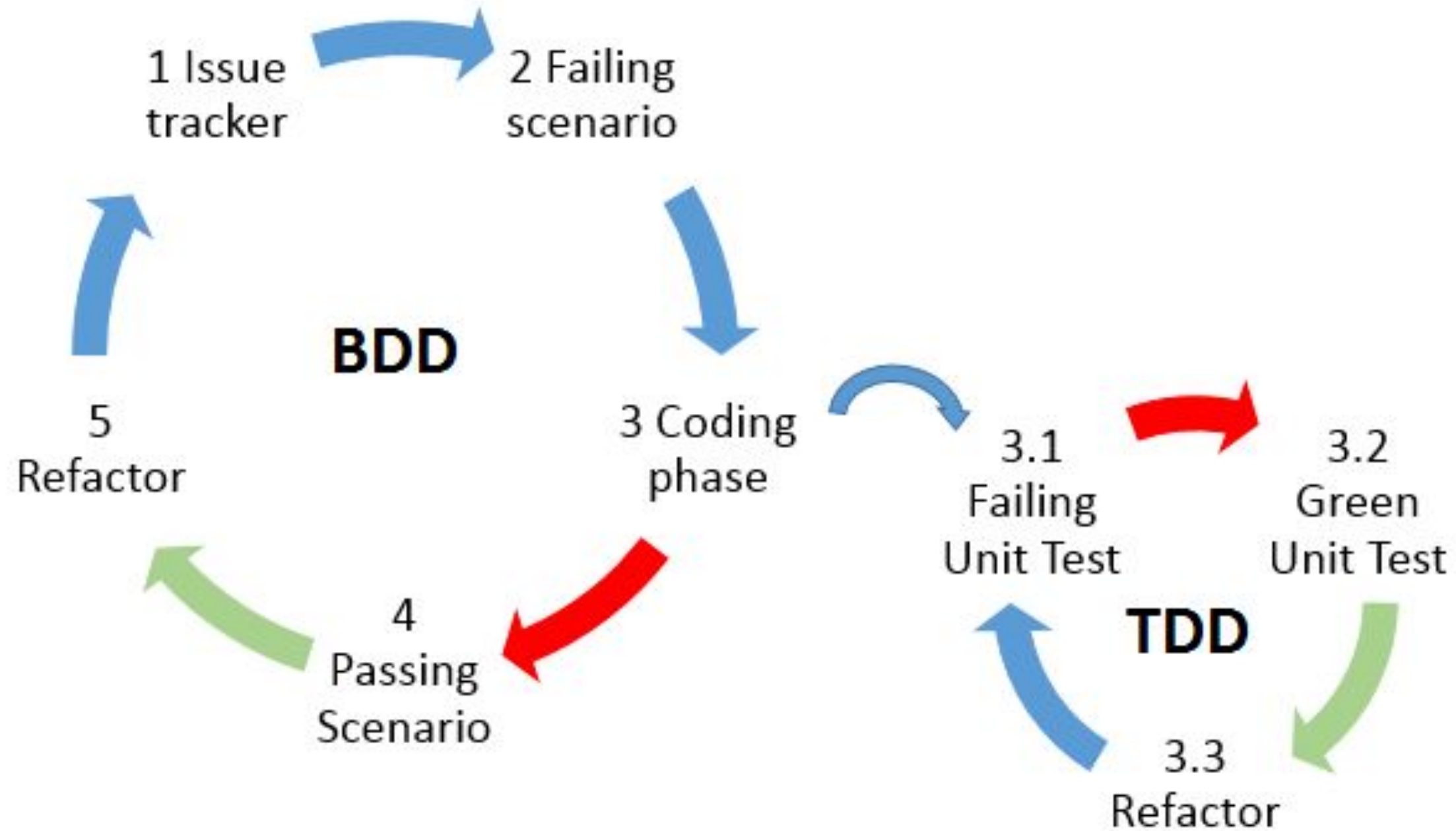
```
Given I am on the home page
When I enter my username "myemail@gmail.com"
And I enter my password "mypassword"
Then I should see my email "myemail@gmail.com" on the dashboard
```

Permite testear totalmente la app desde el punto de vista del usuario sin importar lo que hay detrás

Ejemplo: <https://cucumber.io/docs/guides/10-minute-tutorial/>



■ BDD - Ciclo de desarrollo



■ BDD - Ventajas

- No defines pruebas, defines comportamientos
- Mejora la comunicación entre desarrolladores, testers, usuarios y la dirección
- La curva de aprendizaje es menor que la de TDD
- Como su naturaleza no es técnica, puede llegar a un mayor público
- Encaja perfectamente en las metodologías ágiles que se usan actualmente
- El enfoque de definición ayuda a una aceptación común de las funcionalidades antes de empezar el desarrollo



■ BDD - Inconvenientes

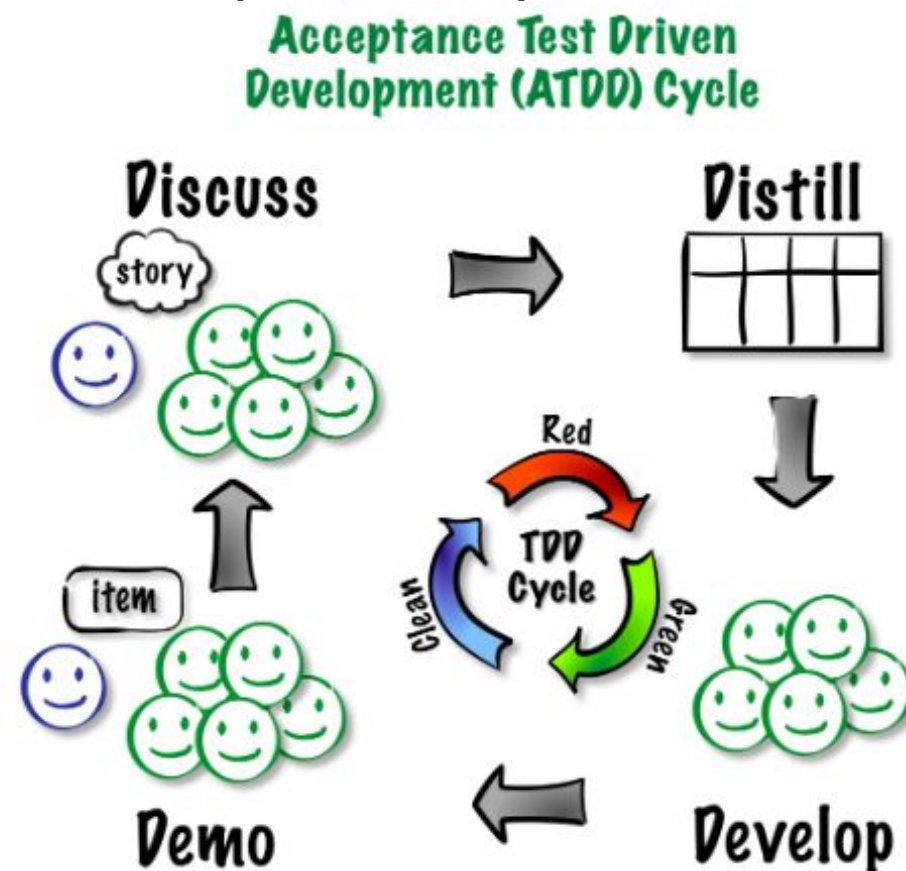
- Errores no identificados en el alcance de la funcionalidad
- Uso con bases de datos
- Es difícil de implementar en el front-end, puesto que está diseñado para el testeo de la lógica de negocio
- Gran inversión de tiempo
- Es necesaria una gran comunicación entre desarrollador y cliente
- Necesidad de tener un equipo de desarrolladores centrados en el trabajo con los clientes



■ ATDD - Qué es

ATDD = “Acceptance Test Driven Development”

- Es más cercano a un proceso que a una actividad
- Según su definición está más cerca de BDD que de TDD
- ATDD busca que lo que se esté haciendo se haga de forma correcta pero también que lo que se hace es lo correcto a hacer



■ ATDD - Ventajas

- No trabajaremos en funciones que finalmente no se van a usar
- Forjaremos un código listo para cambiar si fuera necesario porque su diseño no está limitado por una interfaz de usuario o por el diseño de una base de datos
- Se puede comprobar muy rápido si se están cumpliendo los objetivos o no
- Conocemos en qué punto estamos y cómo se progresa
- El product owner puede revisar los test de aceptación y comprobar cuando se están cumpliendo. Esto mejora la confianza del product owner en sus desarrolladores



GRACIAS

www.keepcoding.io

