Testing

Introducción al desarrollo dirigido por tests

Álvaro Alonso

Pruebas en desarrollo software

- Objetivos de las pruebas (tests)
  - Evaluar la **funcionalidad** de un software
  - Determinar si cumple con los **requisitos** especificados
  - Identificar posibles **defectos**

- Tipos de tests
  - **Funcionales**: comprueban que el software funciona de acuerdo a sus requisitos
  - **No funcionales**: comprueban "cómo de bien" funciona el software
  - **De estructura**: analizan la arquitectura del sistema
  - **De regresión**: analizan la consistencia en nuevas versiones y comprueban que errores detectados no vuelvan a aparecer

Principios del testing

1. Los tests muestran **la presencia** de defectos
2. Los tests **exhaustivos** son imposibles
3. La verificación de la calidad de un sistema debe empezarse **lo antes posible**
4. La mayor cantidad de errores tienden a concentrarse en una parte pequeña del software (***regla 80/20***)
5. Un conjunto de tests que se utilizan repetidamente disminuirá en su eficacia (***paradoja pesticida***)
6. El testing es **dependiente del contexto**
7. La falacia **de ausencia de errores**

Fuente: International Software Testing Qualifications Board

Pruebas funcionales

- **Validan el funcionamiento** del software de acuerdo a sus **requisitos funcionales**
  - Interfaz de usuario
  - APIs
  - Base de Datos
  - Comunicación cliente – servidor

- Tipos de pruebas funcionales
  - Pruebas unitarias
  - Pruebas de componentes
  - Pruebas de integración
  - Pruebas de humo (smoke)
  - Pruebas de aceptación

Pruebas unitarias

- Comprueban la **funcionalidad de un fragmento de código**

- Patron AAA de las pruebas unitarias
  - **Arrange** (organizar)
    - Inicializar objetos y variables que van a utilizarse en el test
  - **Act** (actuar)
    - Ejecutar la acción que se quiere probar llamando al método o función que corresponda
  - **Assert** (confirmar)
    - Comprobar que el resultado de ejecutar la acción se corresponde con el esperado.

Pruebas unitarias

Consideraciones para definir un test unitario

- **Funcionalidad a probar**
  - Ej. la funcionalidad de añadir una entrada de blog
- **Contexto de la prueba**
  - Ej. el usuario no ha iniciado sesión
- **Resultado esperado**
  - Ej. el usuario es redirigido a la página de login

Desarrollo dirigido por tests

- **Test Driven Development (TDD)**
  - Creado por Kent Beck

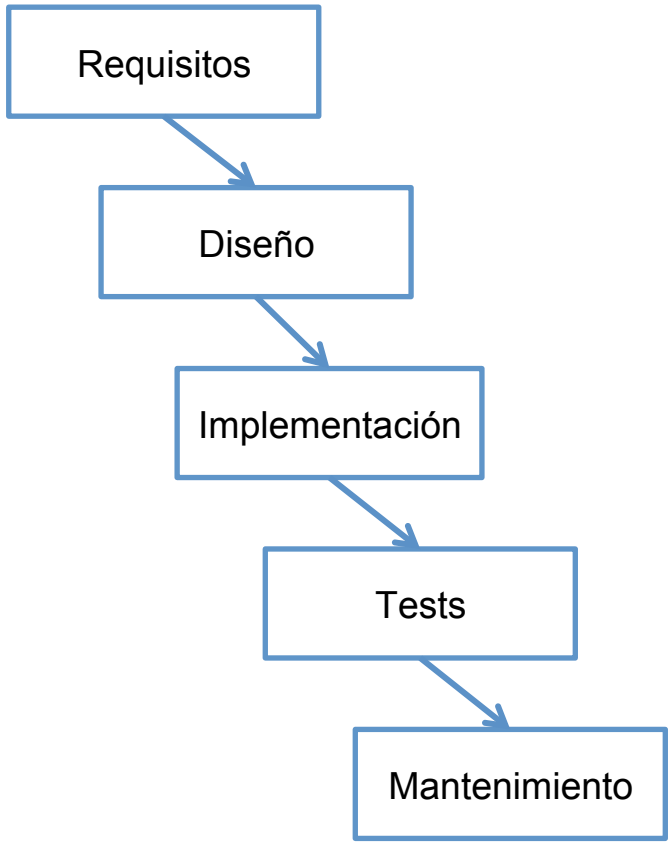
- Práctica de ingeniería de software

1. Definir pruebas unitarias basándose en los requisitos
2. Implementar código para pasar las pruebas
3. Refactorizar código

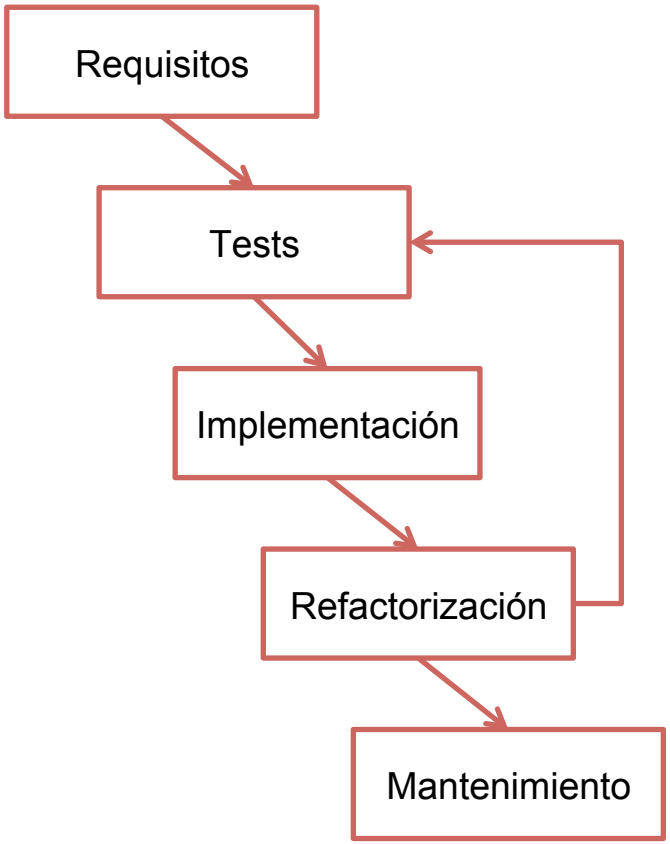
-> Software más robusto y orientado a cumplir requisitos

Desarrollo dirigido por tests

Enfoque tradicional Ingeniería Software

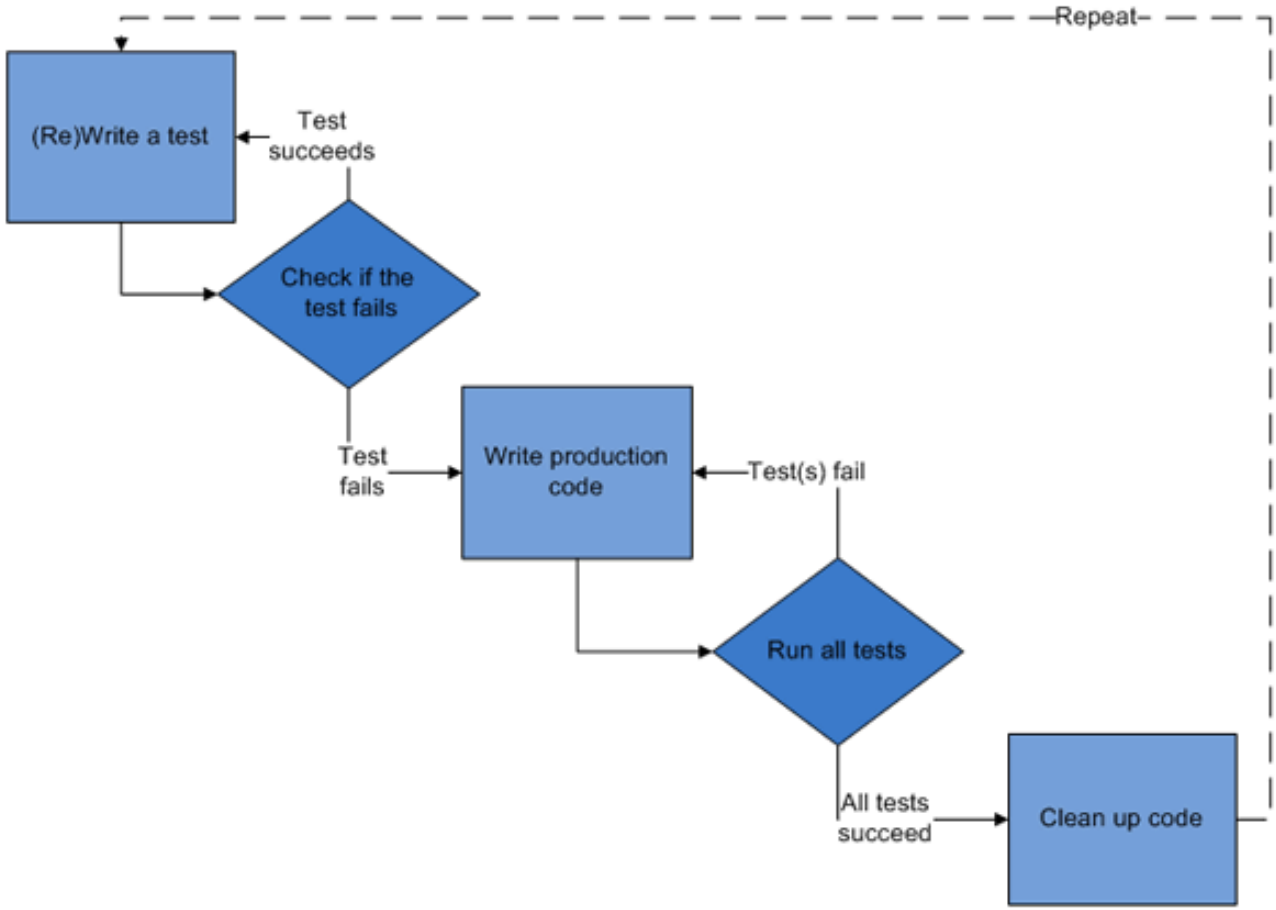


Enfoque Test Driven Development



Desarrollo dirigido por tests

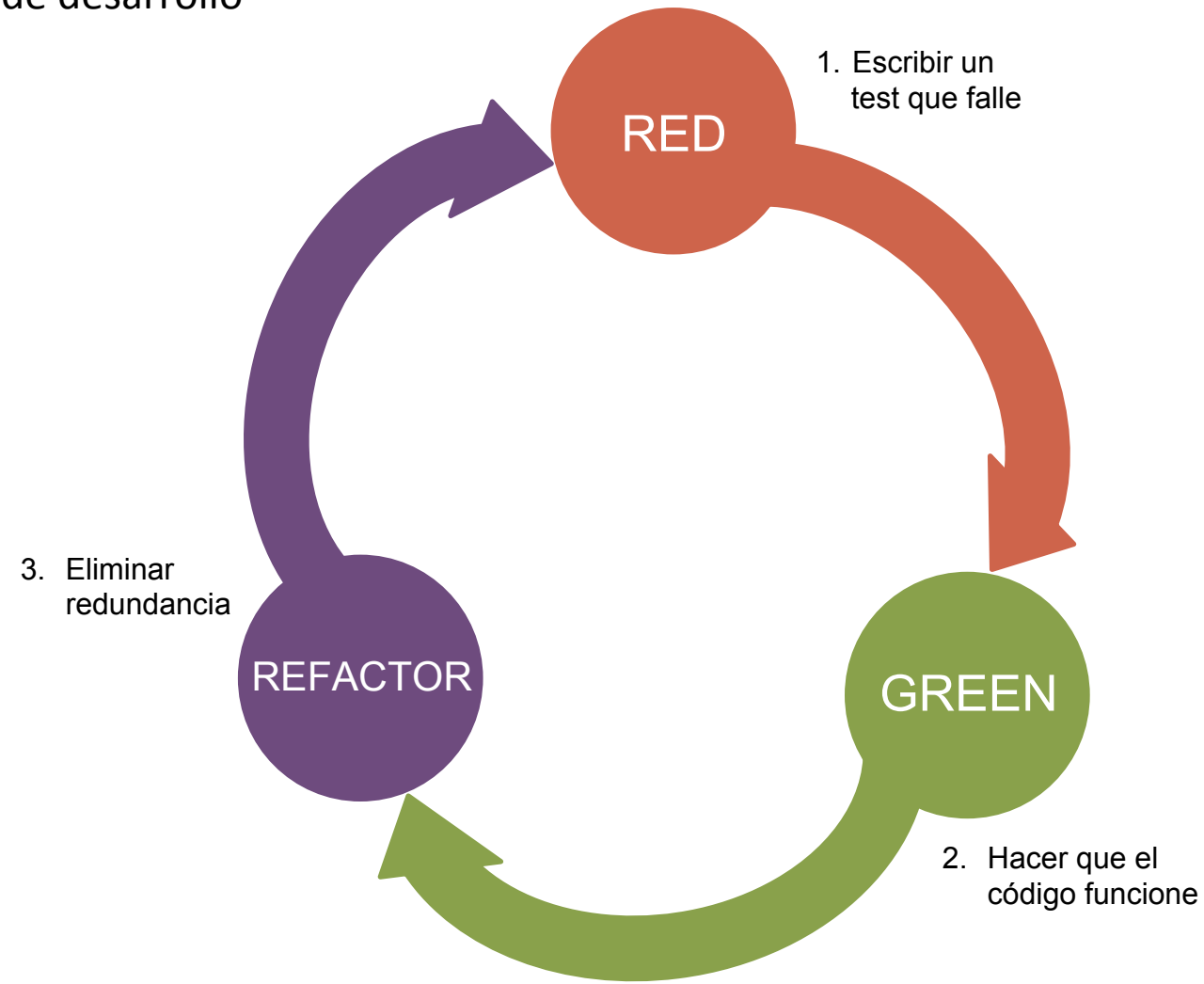
Ciclo de desarrollo



Fuente de la imagen: Wikipedia

Desarrollo dirigido por tests

Ciclo de desarrollo



Desarrollo dirigido por tests

1. **Escribir un test que falle**
   - Descripción del test según una funcionalidad determinada
   - Ejecución del test para comprobar que falla

2. **Hacer que el código funcione**
   - Escribir el mínimo código necesario para que el test pase
     - Sin añadir funcionalidad innecesaria
     - Sin añadir nada que no se vaya a probar
   - Comprobar que el test pasa

3. **Eliminar redundancia**
   - Mejoras en el código sin cambiar funcionalidad
     - Legibilidad, mantenibilidad, complejidad, etc
   - Proceso iterativo

Desarrollo dirigido por tests

Las tres leyes del TDD

1. No debes escribir código de producción hasta que no hayas escrito un test unitario que falle

2. No debes escribir más que un test unitario mínimo que haga que el código falle

3. No debes escribir más código de producción del que sea necesario para pasar el test unitario

Desarrollo dirigido por tests

Ventajas del TDD

- **Mejor diseño** del software
- Código más **limpio** y **estructurado**
- Los tests sirven como **documentación**
- **Detección rápida** de nuevos fallos
- **Depuración** más rápida y eficiente

Desarrollo dirigido por tests

Riesgos del TDD

- **Limitaciones de pruebas unitarias**
  - Pruebas no deterministas
  - Los tests pueden tener errores
- Necesidad de **pruebas de integración**
- Riesgo de “autoengaño”
  - La persona que escribe el test es la que crea el código

TDD en aplicaciones Web

- **Pruebas del lado cliente**
  - Visualización correcta de elementos Web
  - Captura de eventos
  - Envío de formularios
  - Autenticación

- **Pruebas del lado servidor**
  - Rutas de la aplicación
  - Lógica de negocio
  - Renderización de vistas
  - Operaciones que leen/modifican la base de datos

TDD en aplicaciones Web

Herramientas para testing con Javascript y Node.js

- **Mocha**
  - Framework de pruebas de JavaScript que se ejecuta en Node.js.
- **Chai**
  - Librería de aserciones
- **Zombie**
  - Simulador de navegador web para probar lado cliente

Testing

Introducción al desarrollo dirigido por tests

Álvaro Alonso