I.4 Testing

Acceso a Datos

Alejandro Roig Aguilar
alejandro.roig@iesalvarofalomir.org

IES Álvaro Falomir
Curso 2023-2024

Calidad del software

Escribir software libre de defectos es muy difícil.

Una de las mejores formas para tener un grado razonable de certeza de que el software se comporta como es espera es probar su funcionamiento en ciertas circunstancias.

Dijkstra: "Las pruebas de software pueden probar la presencia de errores pero no la ausencia de ellos".

En conclusión, las tareas de prueba permiten verificar que el software que se está creando es correcto y cumple con las especificaciones impuestas por el usuario antes de que llegue a producción.

Pirámide de pruebas

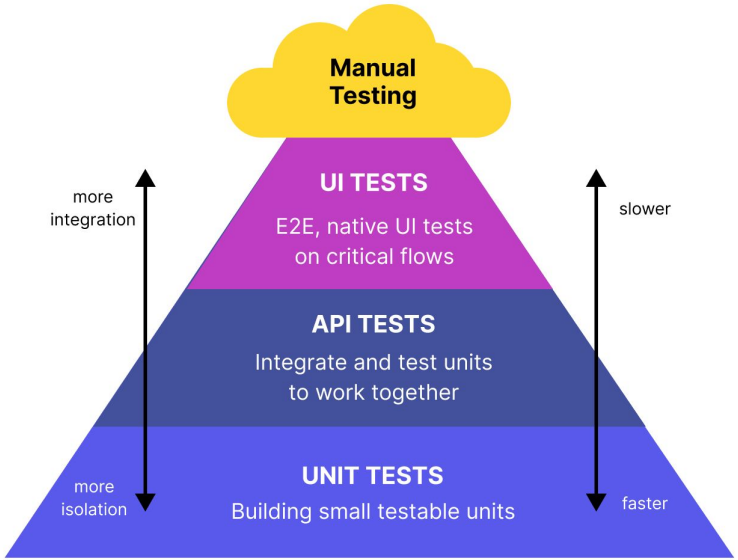
Propone la agrupación de las pruebas en capas y la cantidad relativa de pruebas que se debe crear por cada capa:

- Pruebas E2E. Simulan una experiencia de usuario completa.
- Pruebas de integración. Verifican el funcionamiento entre los componentes que conforman el sistema.
- Pruebas unitarias. Evalúan la funcionalidad de un componente de forma aislada.

Google recomienda una distribución de 70% pruebas unitarias, 20% pruebas de integración, y 10% de pruebas e2e.

Pirámide de pruebas

Si bien las pruebas anteriores son pruebas automatizadas, es necesario tener en cuenta las pruebas exploratorias manuales donde el tester navega por el producto de forma libre.



Técnicas de diseño

El objetivo de una técnica de diseño de pruebas es identificar condiciones de prueba, casos de prueba y datos de prueba.

- Caja blanca. Prueban el código interno de un componente. Destacan la cobertura de sentencias o la cobertura de caminos.
- Caja negra. Evalúan la funcionalidad desde las entradas que recibe y las salidas que produce, sin tener en cuenta su funcionamiento interno. Las estrategias más habituales son las clases de equivalencia y los valores límite.
- No funcionales. Verifican requisitos operativos, como la carga que soporta el producto, si su rendimiento es el correcto o su robustez ante amenazas internas y externas.

Herramientas

Existen multitud de herramientas que facilitan la ejecución de pruebas.

- Pruebas unitarias e integración. Concepto xUnit, que es el nombre genérico de los frameworks de pruebas para diferentes lenguajes, como JUnit en Java, phpUnit en PHP o NUnit en .NET.

Herramientas

- Pruebas E2E. Entre las herramientas para pruebas E2E encontramos Selenium para pruebas de aplicaciones web, Postman para pruebas de API, Appium para aplicaciones móvil y de escritorio, o JMeter para pruebas de rendimiento.

Metodologías

Tradicionalmente, la fase de pruebas aparece después de la implementación.

Metodologías

En la actualidad, es habitual encontrarnos con metodologías TFD (Test First Development), donde el desarrollo de pruebas se realiza antes de la fase de implementación, tras el diseño y análisis.

Kent Beck, autor de *Extreme Programming Explained*: "Un desarrollador no debería escribir una sola línea de código sin antes haber hecho un test que la pruebe."

Metodologías

TDD (Test Driven Development) consiste en el desarrollo de pruebas junto con el análisis, diseño e implementación.

De esta manera, las pruebas guían el desarrollo.

Las pruebas no se realizan de forma manual mientras se desarrolla, sino que se realizan pruebas automáticas.

Metodologías

BDD (Behaviour Driven Development) define pruebas de alto nivel que verifican que el comportamiento del código es correcto desde el punto de vista de los requisitos.

Los requisitos se definen con un formato parecido a un lenguaje natural que es directamente ejecutable, facilitando la comunicación entre todos los implicados, sean técnicos o no.

Es muy utilizado en las actuales metodologías ágiles.

CI/CD

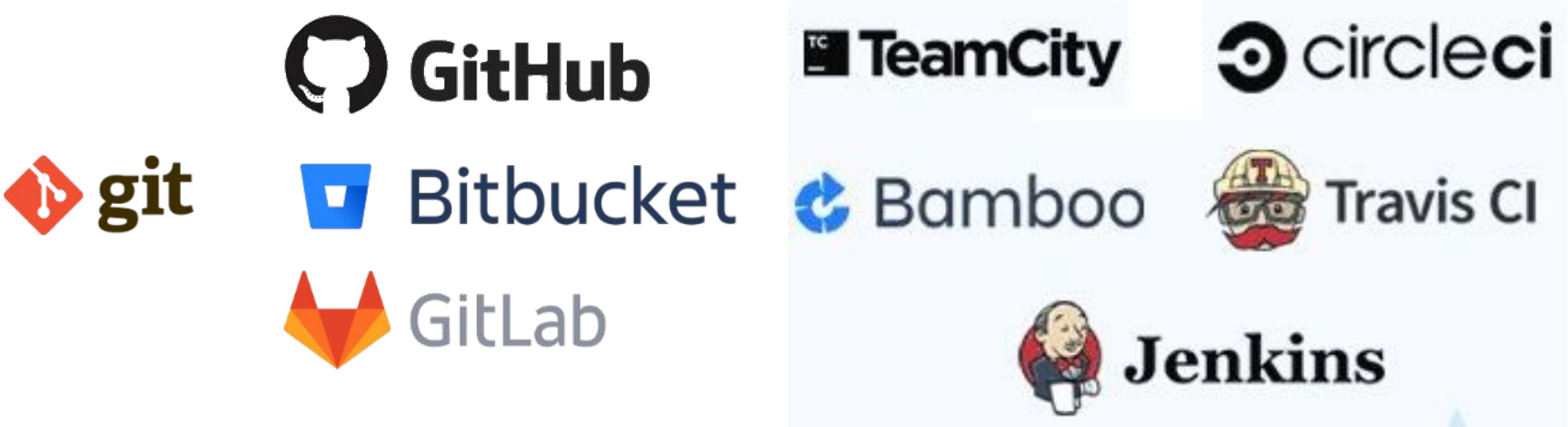
Se considera buena práctica integrar de forma continua (CI) el código desarrollado por los distintos desarrolladores del equipo.

El código se sube a un repositorio compartido, que contiene una herramienta de integración continua que ejecuta las pruebas automáticas sobre el software.

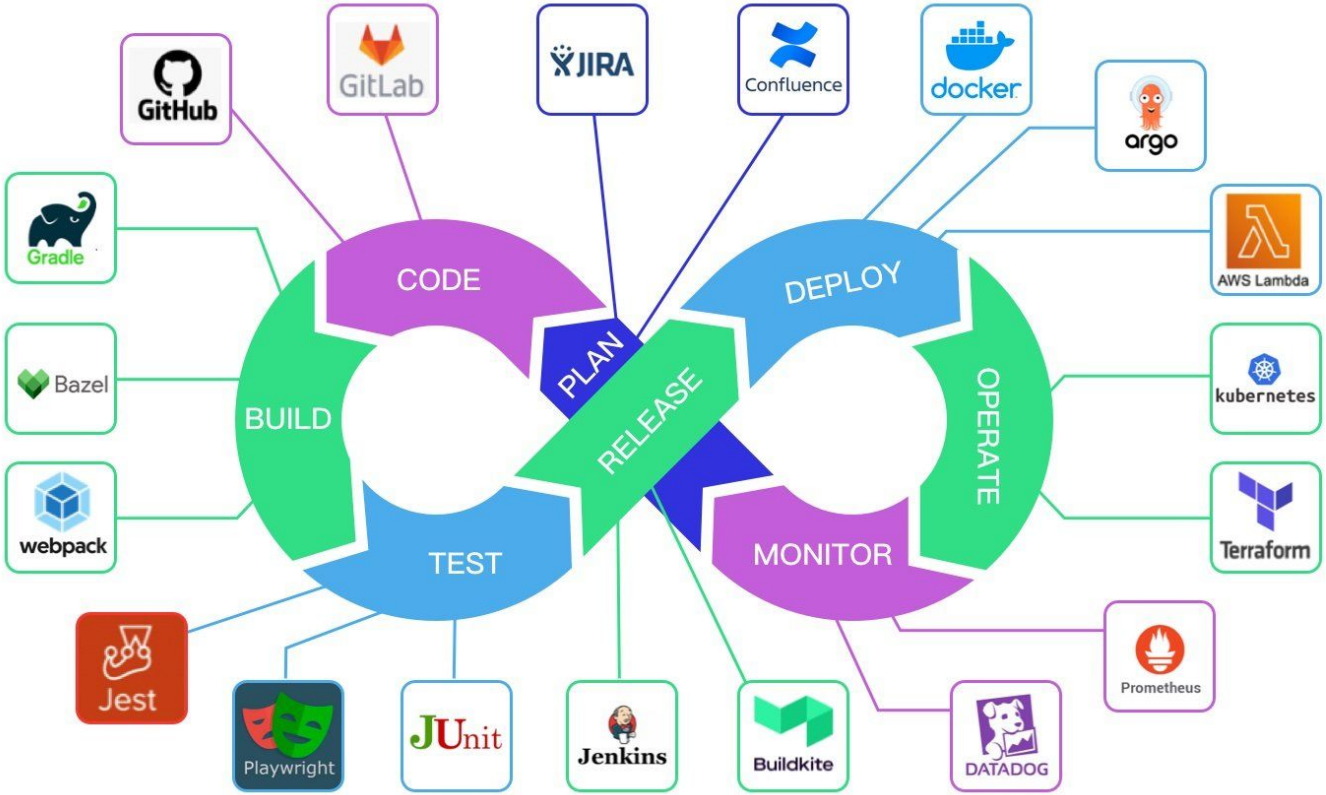
Cualquier problema detectado por CI debe repararse de forma inmediata antes de continuar con el desarrollo.

CI/CD

Como extensión de la CI, el despliegue continuo (CD) automatiza el proceso de despliegue a producción si las pruebas del software son correctas sin requerir de validación humana.



CI/CD


GitHub
GitLab
JIRA
Confluence
docker
argo
Gradle
AWS Lambda
CODE
DEPLOY
PLAN
Bazel
kubernetes
BUILD
RELEASE
OPERATE
webpack
Terraform
TEST
MONITOR
Jest
Prometheus
Playwright
JUnit
Jenkins
Buildkite
DATADOG

JUnit

JUnit es un framework para escribir tests unitarios. Un test tiene la siguiente estructura:

```
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.Assertions;

    class MiTest {
        ...

        @Test        // Anotación para casos de prueba
        void unTest() {
            ...
        }
    }
```

Aserciones

Una aserción es una expresión booleana con el objetivo de comparar y comprobar la corrección en nuestros tests.

Para ello, jUnit dispone de los métodos assert*

Por ejemplo, assertTrue comprueba que la expresión con retorno booleano que le pasamos como argumento es verdadera.

```
int numero = 5;
assertTrue(numero > 0);
```

Disponemos también de los métodos assertFalse o assertEquals.

El uso librerías como Hamcrest, permite potenciar y simplificar los tests. Sin embargo, no es crítica para tener unos tests de calidad.

Ejemplo JUnit

```
@Test
public void testProductoAnyadidoCarro() {
    // Arrange [Escenario]
    Cart carro = new Cart();
    carro.addProduct(new Product("Un libro");

    // Act [Acción]
    String resultado = carro.getProductByName("Un libro");

    // Assert [Resultado]
    assertEquals(resultado.getName(), "Un libro");

}
```