



Universidad de las Fuerzas Armadas ESPE

Departamento de Ciencias de la Computación - DCCO

Carrera de Ingeniería de Software

Pruebas de Software - NRC 22431

Tarea 1: Técnicas estáticas y dinámicas

Luis Ajeajuro Andrade Encalada

/Ing. Luis Alberto Castillo Salinas

03 de Mayo 2025

Índice

1. Introducción.....	1
2. Desarrollo.....	2
2.1. Definición General.....	2
2.2. Técnicas estáticas.....	2
2.3. Técnicas dinámicas.....	4
2.4. Comparación entre ambas técnicas.....	5
2.5. Ejemplo práctico.....	6
3. Conclusiones.....	7
4. Recomendaciones.....	7

Introducción

En el mundo del desarrollo de software, asegurar la calidad y fiabilidad de los productos es una prioridad innegable. Las pruebas de software son un pilar fundamental que sostiene esta garantía, funcionando como un proceso sistemático para identificar defectos, validar funcionalidades y verificar que el software cumpla con las expectativas y requisitos definidos. Este proceso es crítico no solo por la satisfacción del usuario final, sino también para minimizar costos asociados a la corrección de errores y salvaguardar la reputación de los desarrolladores.

Existen dos categorías que se distinguen por su enfoque y momento de aplicación: las técnicas estáticas y las técnicas dinámicas. Mientras las primeras se enfocan en el análisis del código y la documentación sin necesidad de ejecución buscando anomalías en las fases tempranas del proyecto, las segundas se centran en el comportamiento del software en tiempo real, ejecutándolo para observar su respuesta bajo distintas circunstancias. Ambas son indispensables y ofrecen beneficios únicos que, lejos de ser mutuamente excluyentes, se complementan para construir una estrategia de calidad alta.

Este informe tiene como objetivo desglosar y comparar estos enfoques mediante una exploración de sus definiciones, objetivos principales, ejemplos comunes, así como sus fortalezas y limitaciones.

Desarrollo

2.1: Definición General:

¿Qué son las pruebas de Software?

Las pruebas de Software son un proceso fundamental en el ciclo de vida del desarrollo de software que tiene objetivo evaluar la calidad, funcionalidad y rendimiento de una aplicación o sistema. Consiste en ejecutar el software con la intención de encontrar defectos, errores o brechas entre los requisitos esperados y el comportamiento real. Su propósito principal es asegurarse de que el software cumpla con las especificaciones y expectativas del usuario.

¿Cuál es la diferencia entre técnicas estáticas y dinámicas?

La principal diferencia radica en si el código se ejecuta o no durante el proceso de prueba.

- Técnicas Estáticas: Aquellas que no requieren ejecución del código. Se basan en el análisis del código fuente, la documentación. Se realizan antes de la ejecución del código.
- Técnicas Dinámicas: Involucran la ejecución del código del software con el objetivo de observar su comportamiento en tiempo real.

2.2: Técnicas Estáticas

Son un conjunto de métodos utilizados para evaluar la calidad del software sin ejecutar el código. Se enfocan en examinar el código fuente, la arquitectura, los documentos de diseño, los requisitos para detectar errores, inconsistencias, desviaciones de estándares y posibles defectos en etapas tempranas del ciclo de desarrollo. Su objetivo principal es identificar y corregir errores lo antes posible, lo que reduce el costo de las correcciones y mejora la calidad general del software.

ESTILO

Ejemplos comunes:

- Revisión de código: Un examen sistemático del código fuente por parte de uno o más desarrolladores que no son los autores del código.
- Análisis estético: Proceso automatizado de examinar el código fuente o código compilado sin ejecutarlo.
- Inspecciones: Un tipo de revisión de código formal y altamente estructurada. Se documentan los hallazgos y se realizan reuniones para discutir y resolver los problemas.

Ventajas:

- Detección temprana de defectos.
- Mejora de calidad del código.
- Reducción de costos.
- Identificación de vulnerabilidades de seguridad.
- No requiere un entorno de ejecución.

Limitaciones:

- No detecta errores en tiempo de ejecución.
- Falsos positivos.
- Costo inicial de configuración.
- Dependencia de la calidad de las reglas.
- No garantiza la funcionalidad completa.

Herramientas comunes para pruebas estáticas:

- SonarQube: Proporciona paneles de control y métricas.
- ESLint: Herramienta de linting para Javascript y JSX.
- Checkstyle: Escribir código Java de acuerdo a un estándar.
- PyLint: Analizador de código fuente estético para Python.
- FindBugs: Herramienta de análisis estético para Java.

2.3 - Técnicas Dinámicas

Definición y objetivo principal

Implican la ejecución del software en su entorno de prueba para observar su comportamiento, validar su funcionamiento y rendimiento, y detectar errores o defectos. El objetivo principal es verificar que el software cumple con los requisitos funcionales y no funcionales, se comporta como se espera bajo diversas condiciones y es robusto yiable en su operación.

Tipos de pruebas que se consideran dinámicas

- Pruebas Unitarias: Se enfocan en probar componentes individuales del código (funciones, métodos, clases) de forma aislada.
- Pruebas de Integración: Probar la integración entre diferentes módulos o unidades que fueron separadas previamente de forma individual.
- Pruebas de Sistema: Probar el sistema completo e integrado para evaluar su cumplimiento con los requisitos especificados.
- Pruebas de Aceptación: Realizadas para usuarios finales o clientes para verificar que el software satisface las necesidades y requerimientos de negocio.

Ventajas

- Detección de errores de tiempo de ejecución.
- Verificación de requisitos funcionales.
- Medición del rendimiento.
- Evaluación de la experiencia del usuario.
- Cobertura de código.

Limitaciones

- Alto costo y tiempo.
- Detección tardía de defectos.
- Cobertura limitada.
- Dificultad para reproducir errores.
- Requiere entorno de ejecución.

Herramientas para pruebas dinámicas

- JUnit: Pruebas unitarias para Java
- Mocha: Framework para pruebas de JavaScript
- Selenium: Para pruebas de regresión y sistema.
- Pytest: Framework de pruebas para Python
- NUnit: Framework para pruebas en aplicaciones de .NET.

2.4: Comparación entre ambas técnicas

¿En qué momentos de desarrollo se aplican?

- Técnicas estáticas: Etapas tempranas del ciclo de desarrollo.
- Técnicas dinámicas: Etapas posteriores del ciclo de vida de desarrollo de software.

¿Pueden complementarse?

Sí, estas técnicas no solo pueden complementarse, sino que deben hacerlo. Son dos enfoques diferentes que cuando se usan en conjunto ofrecen una mayor cobertura de prueba mucho más completa y efectiva.

Técnicas Estáticas	Técnicas Dinámicas
No se ejecuta el código.	Sí se ejecuta el código.
Aplicación temprana.	Aplicación posterior.
Detección temprana de defectos.	Verificación de funcionalidad, jura de la calidad del código, cumplimiento de estandares.
Errores de sintaxis, violaciones de estandares, problemas de diseño, lógica detectuosa.	Errores de tiempo de ejecución, fallas funcionales, problemas de rendimiento.
Bajo costo de corrección.	Alto costo de corrección.
No requiere entorno de ejecución.	Requiere un entorno de ejecución.
Mayor probabilidad de falsos positivos.	Menor probabilidad de falsos positivos.

2.5 Ejemplo práctico

Considerando el código que se muestra en la figura 1 se define en python el cálculo para el área de un círculo.

```
import math

def calcular_area_circulo(radio):
    """
    Calcula el área de un círculo.

    Args:
        radio (float): El radio del círculo.

    Returns:
        float: El área del círculo.
    """
    if radio < 0:
        return 0 # Probabilidad potencial: área no puede ser negativa, pero ¿debería?
    area = math.pi * radio * radio
    return area
```

Figure 1: Código de ejemplo

Para realizar la prueba estática se puede usar Pylint haciendo uso del comando suponiendo que el archivo se llame "circulo.py"

- 1 pip install pylint
- 2 pylint circulo.py

Para realizar la prueba dinámica se debe crear un archivo de prueba, para este caso se lo llamó "test_circulo.py" definiendo las pruebas como se muestra en la figura 2

```
import pytest
from circulo import calcular_area_circulo
import math

def test_area_circulo_positivo():
    """Prueba que el área calcula correctamente para un radio positivo."""
    radio = 5.0
    expected_area = math.pi * (radio ** 2)
    assert calcular_area_circulo(radio) == pytest.approx(expected_area)

def test_area_circulo_cero():
    """Prueba que el área es cero para un radio de cero."""
    radio = 0.0
    expected_area = 0.0
    assert calcular_area_circulo(radio) == pytest.approx(expected_area)

def test_area_circulo_negativo_retorna_cero():
    """Prueba que un radio negativo retorna cero (según la implementación actual)."""
    radio = -2.0
    expected_area = 0.0
    assert calcular_area_circulo(radio) == pytest.approx(expected_area)
```

Figure 2: Código de prueba

Ejecutando el siguiente script permitirá observar los resultados de la prueba dinámica

- 1 pip install pytest
- 2 pytest test_circulo.py

Conclusiones

La exploración de las técnicas de pruebas estáticas y dinámicas revela que ambas son componentes insustituibles en el ciclo de vida de desarrollo del software. Las pruebas estáticas, al centrarse en el análisis del código fuente y la documentación sin ejecutar, demuestran ser herramientas excepcionalmente valiosas para la detección temprana de defectos. Esto incluye la identificación de vulnerabilidades de seguridad, la garantía de adherencia a estandares de codificación y la mejora general de la calidad del código.

Por otro lado, las pruebas dinámicas son esenciales para validar el comportamiento real del software en un entorno de ejecución. Desde las pruebas unitarias que aseguran la funcionalidad de componentes individuales, hasta las pruebas de sistema y aceptación que verifican la experiencia del usuario final y el cumplimiento de requisitos funcionales y no funcionales. En conjunto, las técnicas estáticas y dinámicas no compiten sino que deben ser evaluadas en conjunto.

Recomendaciones

Para optimizar la estrategia de pruebas en cualquier proyecto de desarrollo, es crucial adoptar un enfoque integral que combine las técnicas estáticas y dinámicas. Implementando revisiones de código y análisis estático desde las primeras etapas del desarrollo, incluso antes de compilar y ejecutar.

Es fundamental establecer un plan de pruebas dinámicas que abarque desde las pruebas unitarias hasta las pruebas de aceptación y rendimiento. La automatización de estas pruebas mediante frameworks y herramientas es clave para garantizar una ejecución consistente y repetible.

Referencias

- [1] R. S. Pressman y B. Lowe, Software Engineering: A Practitioner's Approach, 9th ed. New York, NY, USA: McGraw-Hill Education, 2019.
- [2] I. Sommerville, Software Engineering, 10th ed. Boston, MA, USA: Pearson, 2016.
- [3] International Software Testing Qualifications Board (ISTQB). (2025). ISTQB Certified Tester Foundation Level Syllabus. [En línea]. Disponible en: <https://www.istqb.org/> (Fecha de consulta: 7 de junio de 2025).
- [4] SonarSource. (2025). SonarQube Documentation. [En línea]. Disponible en: <https://docs.sonarqube.org/latest/> (Fecha de consulta: 7 de junio de 2025).
- [5] ESLint. (2025). ESLint Usage Guide. [En línea]. Disponible en: <https://eslint.org/docs/latest/> (Fecha de consulta: 7 de junio de 2025).
- [6] Checkstyle. (2025). Checkstyle Documentation. [En línea]. Disponible en: <https://checkstyle.sourceforge.io/> (Fecha de consulta: 7 de junio de 2025).
- [7] JUnit. (2025). JUnit 5 Documentation. [En línea]. Disponible en: <https://junit.org/junit5/> (Fecha de consulta: 7 de junio de 2025).
- [8] Mocha.js. (2025). Mocha Documentation. [En línea]. Disponible en: <https://mochajs.org/> (Fecha de consulta: 7 de junio de 2025).
- [9] Selenium. (2025). Selenium Documentation. [En línea]. Disponible en: <https://www.selenium.dev/documentation/> (Fecha de consulta: 7 de junio de 2025).
- [10] Pytest. (2025). pytest Documentation. [En línea]. Disponible en: <https://docs.pytest.org/en/stable/> (Fecha de consulta: 7 de junio de 2025).
- [11] Pylint. (2025). Pylint Documentation. [En línea]. Disponible en: <https://pylint.pycqa.org/en/latest/> (Fecha de consulta: 7 de junio de 2025).
- [12] Apache Software Foundation. (2025). Apache JMeter Documentation. [En línea]. Disponible en: <https://jmeter.apache.org/> (Fecha de consulta: 7 de junio de 2025).
- [13] NUnit. (2025). NUnit Documentation. [En línea]. Disponible en: <https://nunit.org/> (Fecha de consulta: 7 de junio de 2025).