Universidad de Concepción

FACULTAD DE INGENIERÍA

DEPARTAMENTO INFORMÁTICA

Fundamentos de testing y aseguramiento de calidad

2025-1

Prof. Geoffrey Hecht

Trabajo: Análisis Estático

Ana Vargas
Sofia Bravo
Esteban Chandía

Concepción, 16 de Julio del 2025

Introducción

En este informe se realizará un análisis del código del proyecto a través de diversas herramientas de análisis estático. El objetivo principal es evaluar la calidad del código, identificar posibles errores de programación, problemas de estilo y vulnerabilidades de seguridad sin necesidad de ejecutar la aplicación. El análisis estático nos permite detectar estos problemas de manera temprana, contribuyendo a mejorar la mantenibilidad, legibilidad y seguridad del proyecto.

Para este análisis se seleccionaron 3 herramientas para proyectos en Python. Cada una de estas herramientas cumple una función distinta y específica dentro del proceso de revisión del código, permitiendo obtener una visión completa de las posibles mejoras necesarias.

Este informe se compondrá de diversas secciones en las que se abordarán los siguientes temas: breve descripción de las herramientas, un resumen de los resultados obtenidos y una reflexión final respecto al uso y relevancia de estas herramientas y el análisis estático en general en el contexto de proyectos de este tipo.

Breve descripción de las herramientas utilizadas

La primera herramienta que utilizamos fue Pylint. Pylint es una herramienta de análisis estático de código fuente para Python la cual su función principal es revisar la calidad del código, detectando errores de sintaxis, violaciones a convenciones de estilo (PEP8), código no utilizado, nombres de variables poco descriptivos y posibles errores lógicos. Además, ofrece una puntuación global del código sobre 10, lo que permite tener una métrica cuantitativa para evaluar mejoras tras refactorizaciones.

Luego tenemos Flake8. Flake8 es una herramienta ligera de análisis estático enfocada principalmente en verificar el cumplimiento de las guías de estilo PEP8 y errores simples de programación. Flake8 integra tres herramientas en una: pyflakes para detectar errores simples, mccabe para medir la complejidad ciclomática, y pep8 para verificar el estilo. Es muy útil para mantener un código limpio, legible y homogéneo.

Finalmente, utilizamos Bandit, esta herramienta se especializa en el análisis estático de seguridad en proyectos Python. Su función es detectar patrones de código que pueden resultar en vulnerabilidades o riesgos de seguridad, como el uso de funciones peligrosas (eval, exec), manejo incorrecto de contraseñas o configuraciones inseguras. Bandit analiza todo el proyecto de manera recursiva y entrega un informe detallado con el nivel de severidad de cada hallazgo.

Resultados del Análisis

1. Pylint

- **Puntaje promedio del proyecto:** 5,65/10
- **Principales advertencias:**
    - Variables no utilizadas.
    - Nombres de variables no descriptivos.
    - Métodos con demasiadas ramas (too-many-branches).
    - Uso de excepciones generales.

Pylint resultó muy útil para identificar áreas del código con baja calidad y mejorar la estructura general del proyecto. También aportó una métrica clara para monitorear avances tras refactorizaciones.

2. Flake8

- **Errores detectados:** 208
- **Principales observaciones:**
    - Saltos de línea innecesarios.
    - Espacios en blanco extra.
    - Líneas con más de 79 caracteres.
    - Indentación inconsistente.
    - Variables declaradas pero no utilizadas.

Flake8 permitió mantener un estilo de código limpio y coherente. Aunque no detecta errores lógicos complejos, es muy útil para asegurar que el código se adhiera a los estándares del lenguaje y sea fácil de leer y mantener.

3. Bandit

- **Líneas analizadas:** 1423
- **Problemas detectados:** 3 (todos de baja severidad)

Detalles:

- **Hardcoded password:**
    - Archivo: app/config.py

    - Línea: SECRET_KEY = 'secret-key-goes-here'
    - Clasificación: Severidad baja, confianza media.
- **Uso de módulo riesgoso (subprocess):**
    - Archivo: funciones_archivo/manejoMaven.py
    - Clasificación: Severidad baja, confianza alta.
- **Llamada a subprocess sin shell=False:**
    - Riesgo potencial si se pasa entrada no validada al comando.

Aunque los problemas detectados no son críticos, es importante corregirlos para evitar futuras vulnerabilidades. En entornos de producción, errores de este tipo pueden ser explotables.

Retrospectiva y Relevancia del Uso de Herramientas

El uso de herramientas de análisis estático en el proyecto aportó beneficios significativos en tres aspectos clave:

- **Calidad del código:** Pylint y Flake8 permiten mejorar la estructura, estilo y legibilidad del código, haciendo que sea más fácil de mantener y escalar una vez aplicadas las modificaciones.
- **Seguridad:** Bandit reveló problemas simples pero relevantes que podrían causar vulnerabilidades si no se abordan.
- **Automatización del control de calidad:** Estas herramientas pueden integrarse fácilmente en pipelines de integración continua (CI), ayudando a mantener estándares consistentes a lo largo del desarrollo.

Conclusión

El análisis realizado con Pylint, Flake8 y Bandit demostró ser una práctica efectiva para detectar y corregir errores en etapas tempranas del desarrollo. Estas herramientas mejoran la calidad, seguridad y mantenibilidad del código, y su uso debería ser considerado una práctica estándar en proyectos Python de cualquier escala.