Herramientas Open Source para Aseguramiento de la Calidad de Aplicaciones Móviles

Dr. Danilo Domínguez

Acerca de mí

- Danilo Domínguez Pérez
- Estudios
  - Egresado de la UTP - Ing. en Sistemas y Computación
  - Maestría en Ciencias Computacionales del Rochester Institute of Technology
  - Doctorado en Ciencias Computacionales de Iowa State University
    - Investigación en Análisis y Testing de Aplicaciones Móviles
- Profesor tiempo parcial en la UTP
- Miembro de FLOSSpa
- Ingeniero de Software Móvil

@danilo04

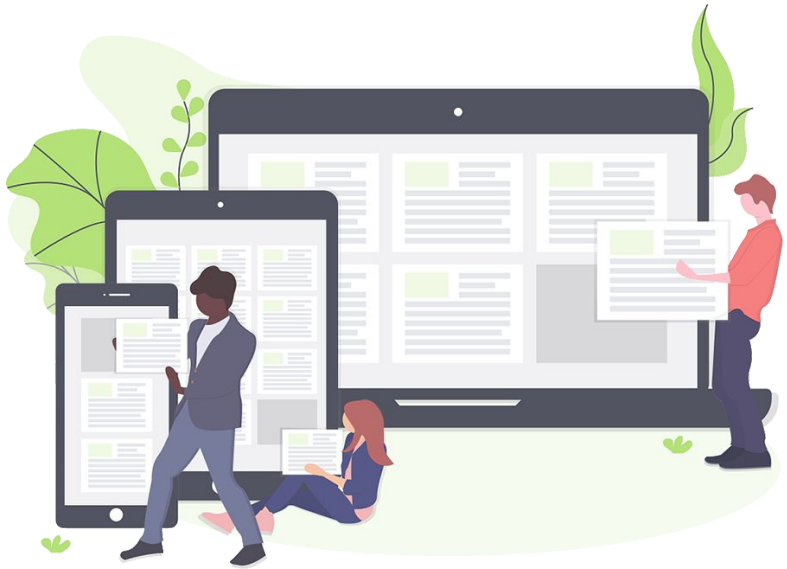
@danilo04

@danilo04.bsky.social

https://www.linkedin.com/in/danilo-dominguez-perez

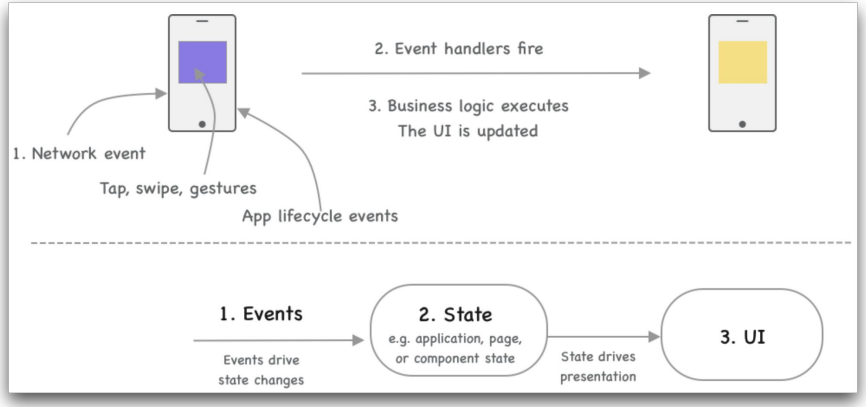
Retos de Desarrollo de Apps Móviles

- Distribución del binario del app
- Cambios de conexión, pobre conexión en algunos casos
- Crashes
- Manejo de estado
- Etc.



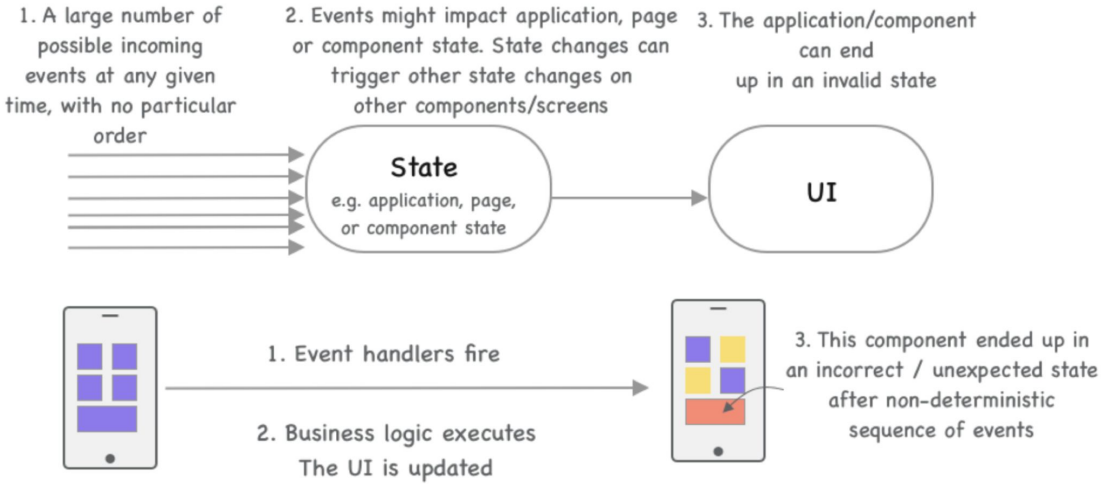
Manejo de Estado

- Ciclo de vida del app no es un problema en web y backend
- Transiciones de ciclo de vida como cuando la app es enviada al background o regresa al foreground
- El OS recupera memoria por lo que en algunos casos termina el app por completo



Manejo de Estado y Bugs

- La mayoría de bugs y crashes pasan por combinaciones de eventos no esperadas o no probadas



Versiones Antiguas

- Pueden mantenerse en uso por largo tiempo
- Hay apps que tienen ventanas de rollouts para realizar actualizaciones
- e.g. Whatsapp, Facebook Messenger.
- Algunos usuarios deshabilitan actualizaciones automáticas
- Otros tienen teléfonos viejos o versiones antiguas de OS que no permiten actualizar a nuevas versiones del app

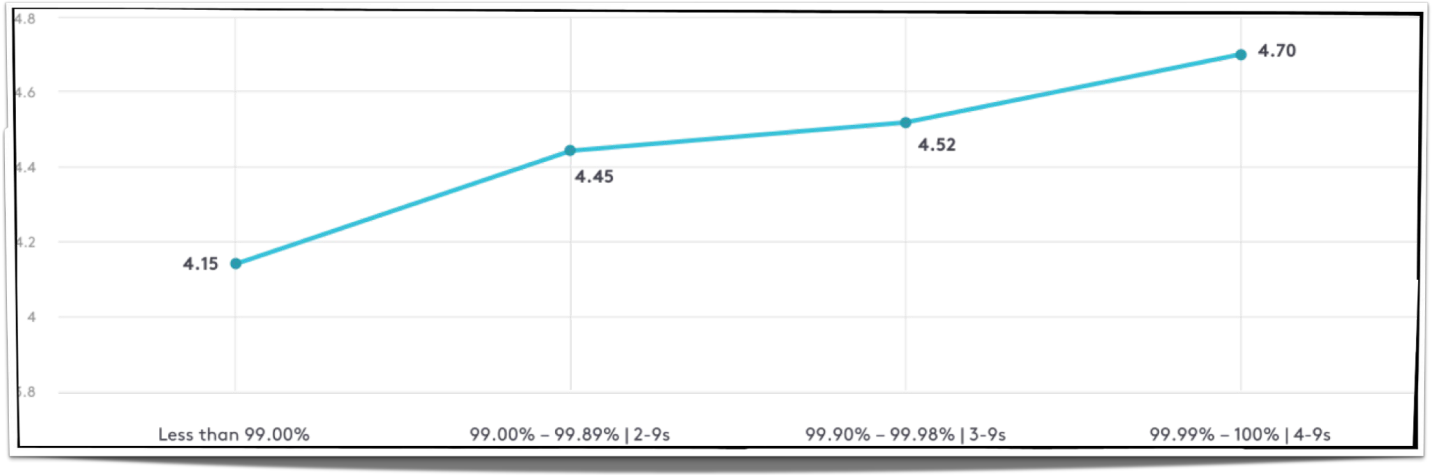
Crashes son los bugs más notables con gran impacto en el negocio

Resultados de Crashes

- User churn
- Usuarios pueden quedar frustrados con el app y buscar una alternativa
- Pueden dejar reviews bajos, afectando la decisión de futuros usuarios
- Crashes también pasan en servicios backend (errores 5XX)


~99%
estabilidad
correlaciona > 4.7 ratings

Mayor estabilidad ≃ Mejores Ratings



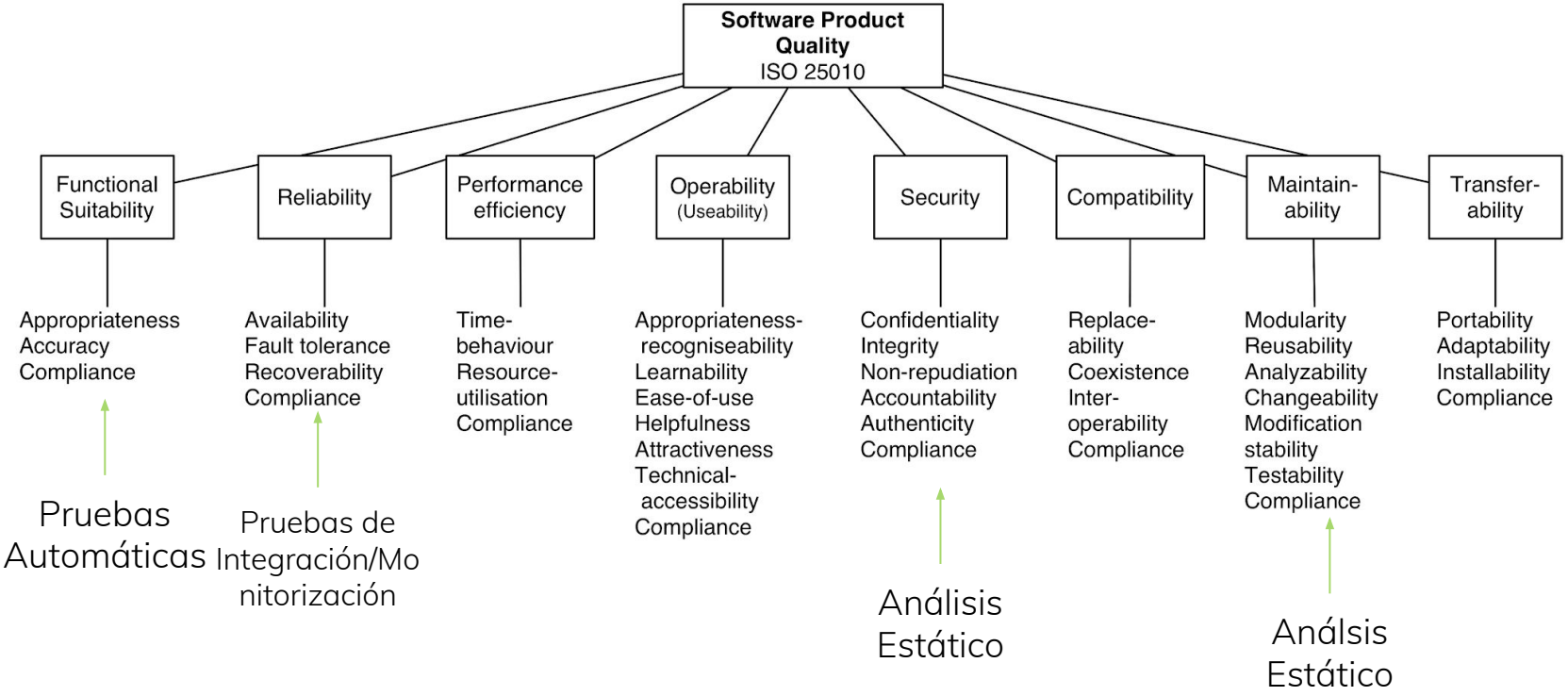
Application Stability Index: Characteristics of Leading Mobile Apps por BugSnag

Aseguramiento de la Calidad en Software Ágil

Pruebas Automáticas

¿Es suficiente?

Atributos de Calidad



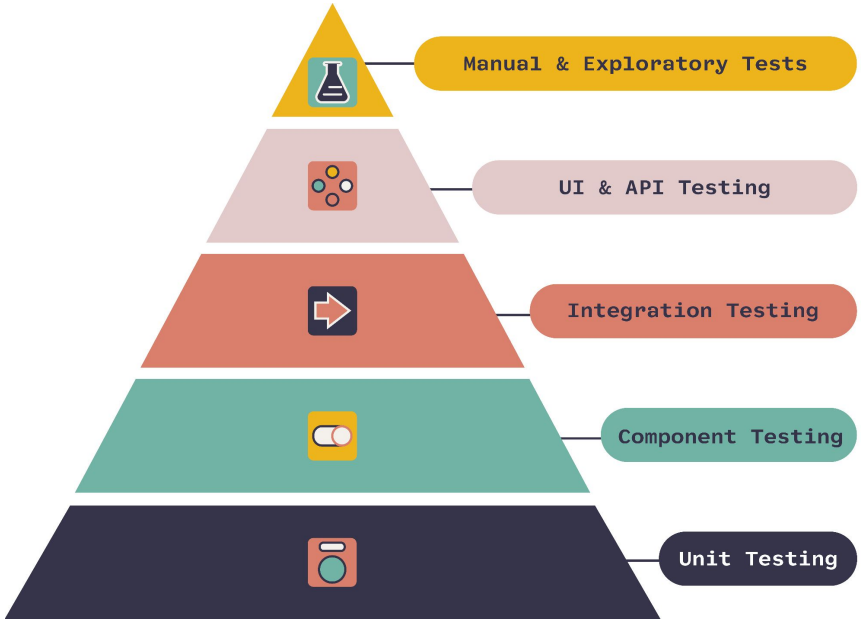
Confiabilidad

01	Pruebas Automáticas	• Pruebas Unitarias • Pruebas de Integración • Pruebas de UI
02	Análisis Estático	• Patrones de código con alta probabilidad de introducir fallas
03	Monitoreo	• Monitorear fallas (crashes)

Pruebas Automatizadas

Herramientas para la implementación de pruebas

Pirámides de Pruebas de Software



https://www.onpathtesting.com/blog/qa-testers-what-is-the-agile-testing-pyramid

Unit Tests

- Para Unit Tests podemos utilizar `xUnit` frameworks o similar
- Para Android `jUnit`
- Para Flutter, `flutter_test`
- Para React Native `Jest`

https://junit.org/junit5/

https://docs.flutter.dev/cookbook/testing/unit/introduction

https://jestjs.io/

Todos son Open Source Frameworks

Property-Based Testing?

¿Qué es property-based testing?

- Basado en propiedades: verificar que las declaraciones o invariantes sean ciertas para un ejemplo.
- Generativo: generar ejemplos automáticamente.

```
[4, 2, 6, 1] => sort() => [1, 2, 4, 6]
for i in result: result[i] <= result[i+1]
```

¿Qué es property-based testing?

En vez de

```
def test_sort_on_example():
    assert sorted([3, 5, 18, 2, -4]) ==
[-4, 2, 3, 5, 18]
```

Describimos qué propiedad debe ser cierta

```
from hypothesis import given
from hypothesis.strategies import lists, integers
@given(lists(integers()))
def test_sort_produces_correct_order(a_list):
    sorted_lst = sorted(a_list)
    for ix in range(len(sorted_lst) - 1):
        assert sorted_lst[ix] <= sorted_lst[ix + 1]
```

¿Por qué necesito un framework?

- ¿Cuál es el problema de `import random` o `Math.random()`?

Un buen framework provee:

- La habilidad de reducir casos fallidos

[--3892949711536414768617173852449709074 53L, 988, -88, 1665, 1649, -3892949711536414768617173852450472658 93L, ..., -938, -3892949711536414768617173852447221914 00L, -3892949711536414768617173852453150666 55L, 751]

Se convierte en

[0, -1]

En especial para pruebas de edge cases

eg, 0, +/-Inf, [], ''

React Native

- Herramientas como:
  - Fast-Check
  - JSVerify

```
import fc from 'fast-check';
import {
  encrypt,
  decrypt
}
from './cryptoUtils';

describe('Encrypt/Decrypt', () =>{
  it('descifrar después de cifrar devuelve la entrada original', () =>{
    const key = 'secretKey123';
    fc.assert(fc.property(fc.string(), (input) =>{
      const encrypted = encrypt(input, key);
      const decrypted = decrypt(encrypted, key);
      return decrypted === input;
    }));
  });
});
```

Android

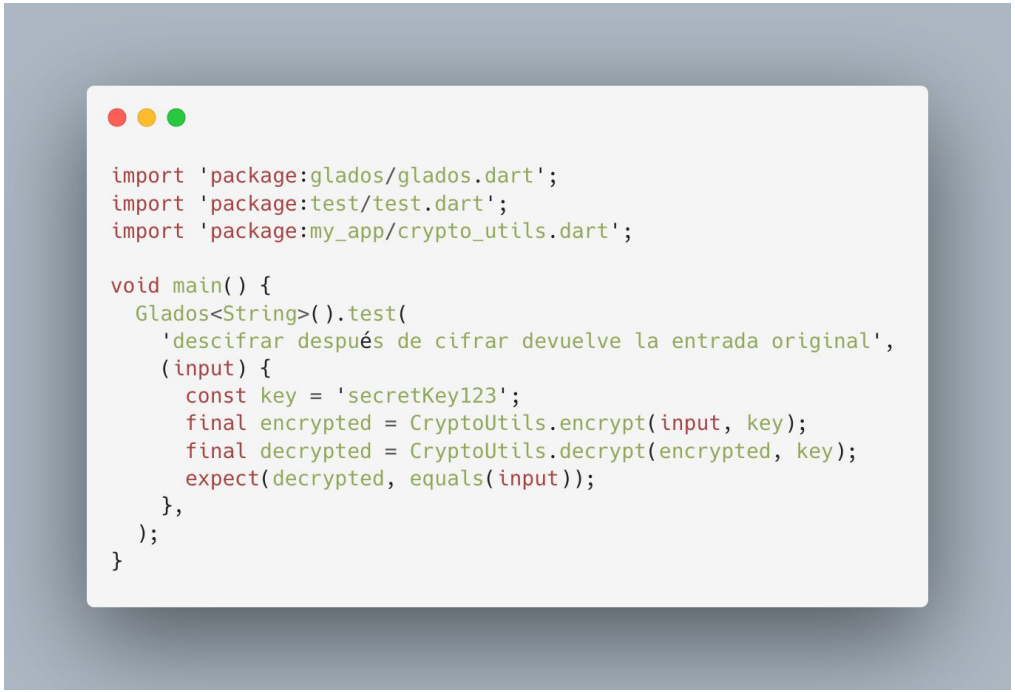
- Herramientas como:
  - Jqwik
  - junit-quickcheck

```
import net.jqwik.api.*;
import static org.junit.jupiter.api.Assertions.assertEquals;

class CryptoUtilsTest {
    @Property
    void decryptAfterEncryptReturnsOriginal(@ForAll String input) {
        String key = "secretKey123";
        String encrypted = CryptoUtils.encrypt(input, key);
        String decrypted = CryptoUtils.decrypt(encrypted, key);
        assertEquals(input, decrypted);
    }
}
```

Flutter

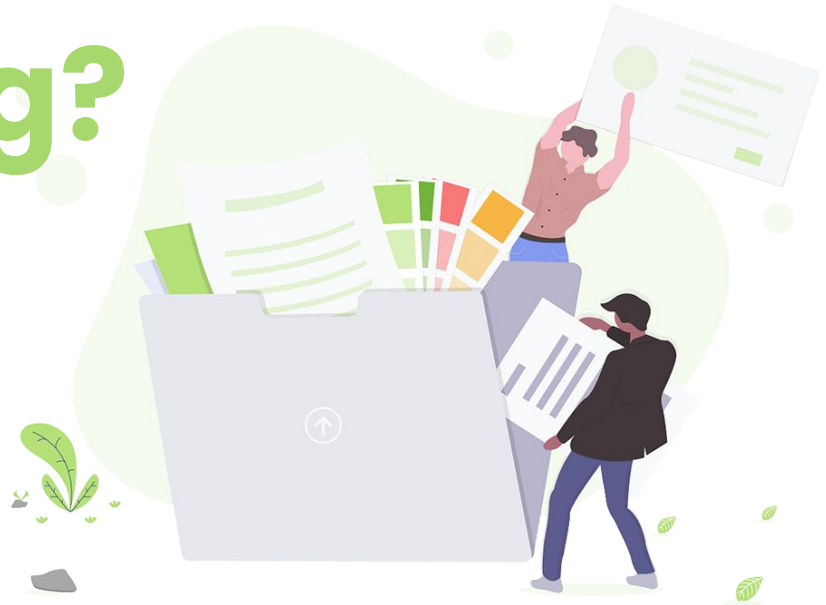
- Herramientas como:
  - Dart-check
  - Glados



```dart
import 'package:glados/glados.dart';
import 'package:test/test.dart';
import 'package:my_app/crypto_utils.dart';

void main() {
  Glados<String>().test(
    'descifrar después de cifrar devuelve la entrada original',
    (input) {
      const key = 'secretKey123';
      final encrypted = CryptoUtils.encrypt(input, key);
      final decrypted = CryptoUtils.decrypt(encrypted, key);
      expect(decrypted, equals(input));
    },
  );
}
```

UI y E2E Testing?



Appium

- Herramienta de código abierto para automatización de pruebas en aplicaciones móviles.
- Soporta Android, iOS y aplicaciones híbridas/nativas.
- Basado en el protocolo WebDriver (compatible con Selenium).
- Pruebas funcionales:
  - Validar flujos de usuario (inicio de sesión, navegación).
  - Verificar respuestas de la UI ante entradas específicas.
- Pruebas de regresión:
  - Automatizar pruebas repetitivas para nuevas versiones.

https://github.com/appium/appium

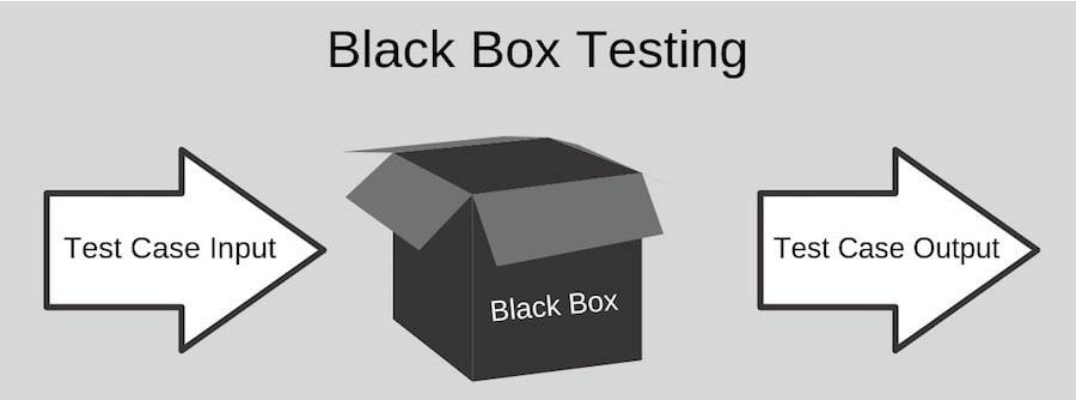
Black-box Testing?

Generación Automática de Entradas

- En testing un reto importante es la generación de test input
- La generación automática ayuda a liberar el peso del proceso manual
- Tipos de generadores de input
    - Aleatorios (Random)
    - Basado en Modelos (Model-based)
    - Search-based
    - Symbolic Execution

Black-box Testing

- Conocimiento de estructura interna del software no es requerido
- Normalmente un diseñador selecciona test inputs
- El output es verificado con la ayuda de **test oracles**



Monkey

- Herramienta de línea de comandos
- Generación automática de test inputs (eventos GUI y del sistema)
- Eventos pseudo-aleatorios
- Black-Box Testing
- Puede ejecutarse en emuladores o dispositivos reales

¿Cómo Ejecutar Monkey?

```
adb shell monkey -p <paquete> <numero-eventos>
```

Parámetro	Descripción
--help	Muestra ayuda incluyenda la lista completa de parámetros
-v	Modo verboso
-s <seed>	Semilla para el generador seudo-aleatorio de eventos
--throttle <milliseconds>	Inserta un retraso fijo entre cada evento
...	...

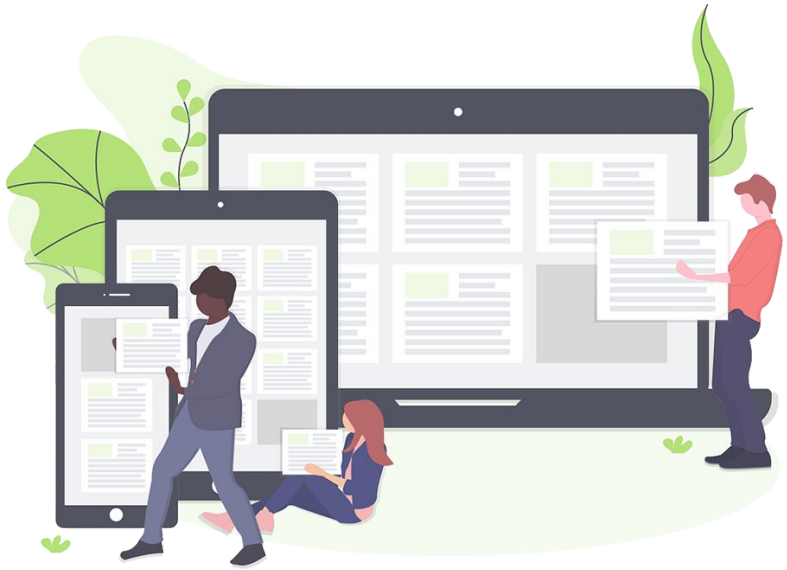
Mantenibilidad

Herramientas para Mejorar/Mantener Mantenibilidad

01	Code Smells y Formatting	• Patrones de código con alta probabilidad de introducir fallas • Patrones de código que no siguen los estándar de calidad de la empresa
02	Dependencias	• Actualización de Dependencies • Revisión de nuevas dependencies y el tamaño del app

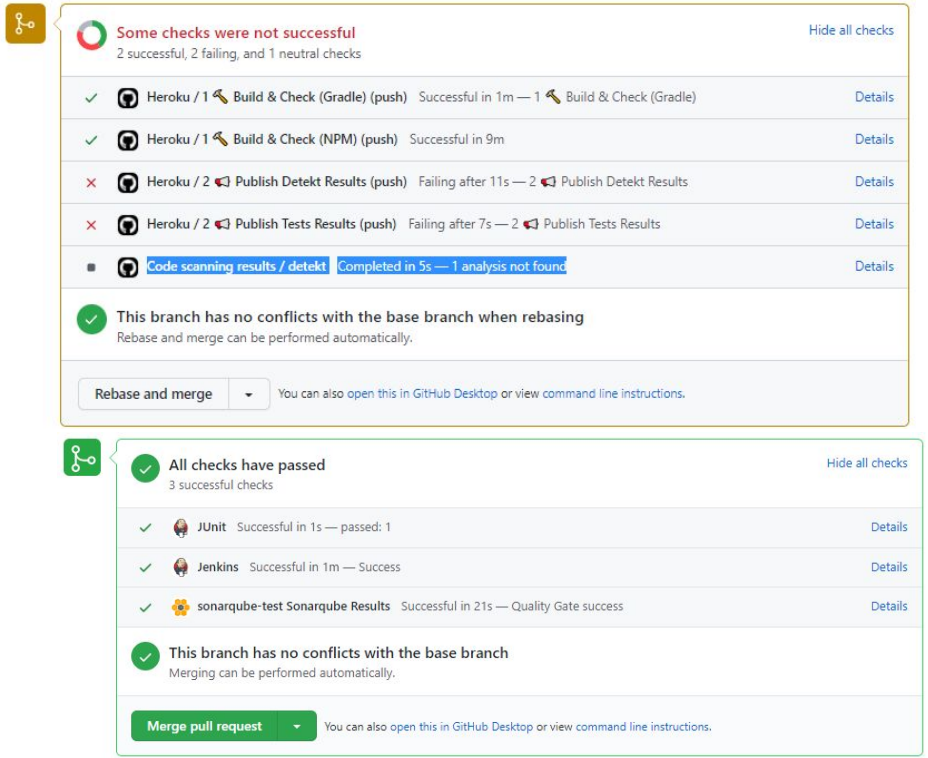
Análisis Estático

Herramientas de análisis estático también nos ayudan a mantener un codebase uniforme



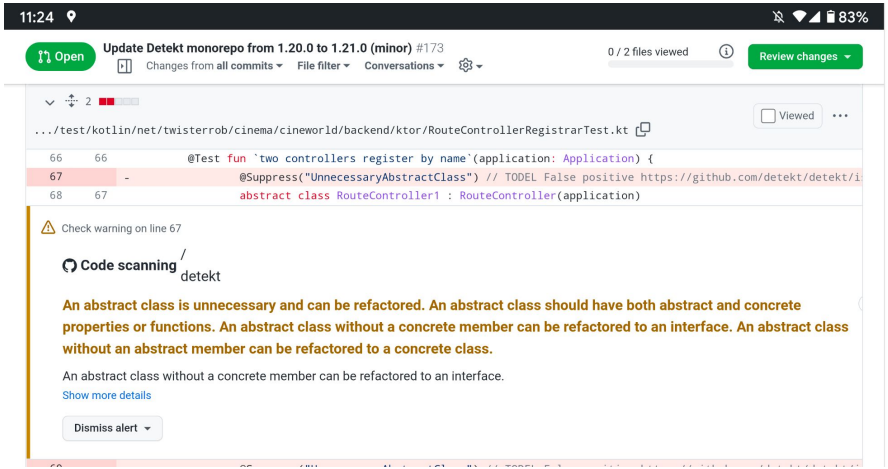
Herramientas de Análisis Estático

- Para Kotlin existen varias herramientas como
  - Detekt
  - Klint
  - AndroidLint
  - SonarQube
- Para React Native
  - ESLint para linting y TypeScript para verificación de tipos.
- Se puede integrar con el sistema de integración continua



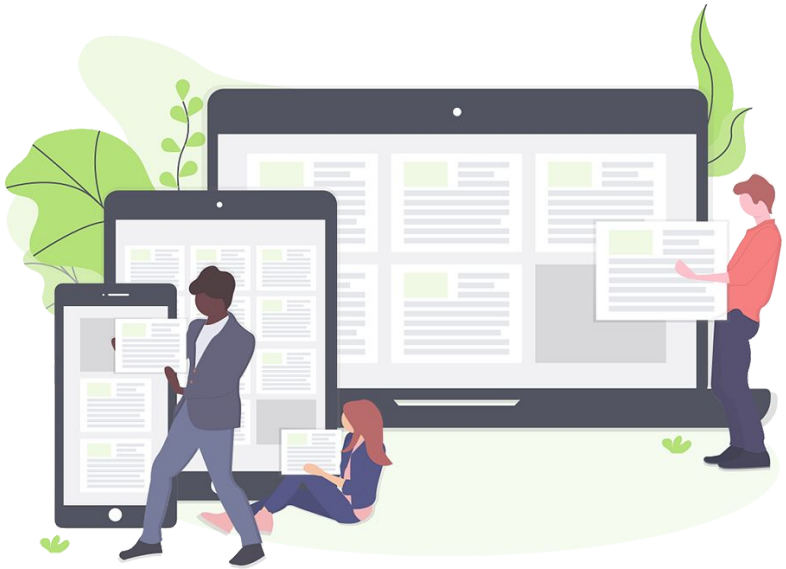
Detección de Code Smells y Antipatrones en CI

- Herramientas como Detekt y EsLint pueden reportar warnings durante CI
- También se ejecutan en los ambientes locales de los desarrolladores
- Tienen la capacidad de aceptar nuevas reglas escritas exclusivas para el equipo



Dependencias

¿Qué debemos tomar en cuenta acerca de nuestras dependencies?

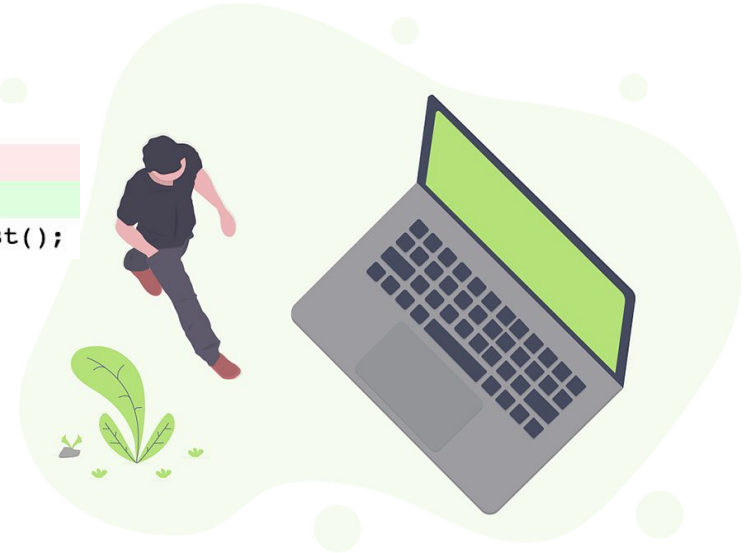


Cambios Ocultos en Solicitudes de Cambios

- La programación se ocupa con frecuencia de cambios visibles

```
.sorted((a, b) -> Long.compare(b.created_at(), a.created_at()))
.flatMap(pendingRequest -> {
.concatMap(pendingRequest -> {
  InitiatePaymentRequest paymentRequest = pendingRequest.payment_request();
```

- Hay bastantes cambios ocultos asociados:
  - dependencias transitivas,
  - código generado
  - archivos de manifiesto



¿Cómo afectan los cambios ocultos?

- Los cambios de dependencia transitivos o los cambios de código generados
  - pueden inflar el binario resultante o
  - reducir el rendimiento
- Estos problemas a menudo se encuentran mucho más tarde en el proceso de liberación, donde se requiere una investigación para encontrar la causa

Kochiku Robot Readonly

Branch	SHA	APK Size	Method Count
`master` (ancestor)	`edcd6fc092f`	7572479	48964
`jw/kot`	`11b6f07deb2`	7572251	48961
		-228	-3

Dependency graph changes:

```
@@ -125,2 +125,3 @@
-|    \--- org.jetbrains.kotlin:kotlin-stdlib:1.0.4 -> 1.1-M03
-|         \--- org.jetbrains.kotlin:kotlin-runtime:1.1-M03
+|    \--- org.jetbrains.kotlin:kotlin-stdlib:1.0.4 -> 1.1-M04
+|         \--- org.jetbrains.kotlin:kotlin-runtime:1.1-M04
+|              \--- org.jetbrains:annotations:13.0
@@ -145,2 +146 @@
-+--- org.jetbrains.kotlin:kotlin-stdlib:1.1-M03 (*)
-+--- org.jetbrains.kotlin:kotlin-runtime:1.1-M03
++--- org.jetbrains.kotlin:kotlin-stdlib:1.1-M04 (*)
```

¿Cómo afectan los cambios ocultos?

- **Dependency Tree Diff** es una herramienta de diff que encuentra cambios en dependencies de Gradle
- Cambios en el archivo de manifiesto
- En cada compilación de CI, se calculan datos interesantes como
  - el tamaño binario
  - el recuento de métodos (una métrica importante para las aplicaciones de Android)

Kochiku Robot Readonly

Branch	SHA	APK Size	Method Count
`master` (ancestor)	`d343d42ae53`	9203505	49320
`jw/2016-06-06/add-whorlwind`	`45af5632e94`	9218183	49439
		+14678	+119

(Note: ancestor commit is 2 behind `master`. Rebase for accurate stats.)

Dependency graph changes:

```
@@ -173,0 +173,1 @@
++--- com.squareup.whorlwind:whorlwind:1.0.1
```

Merged manifest changes:

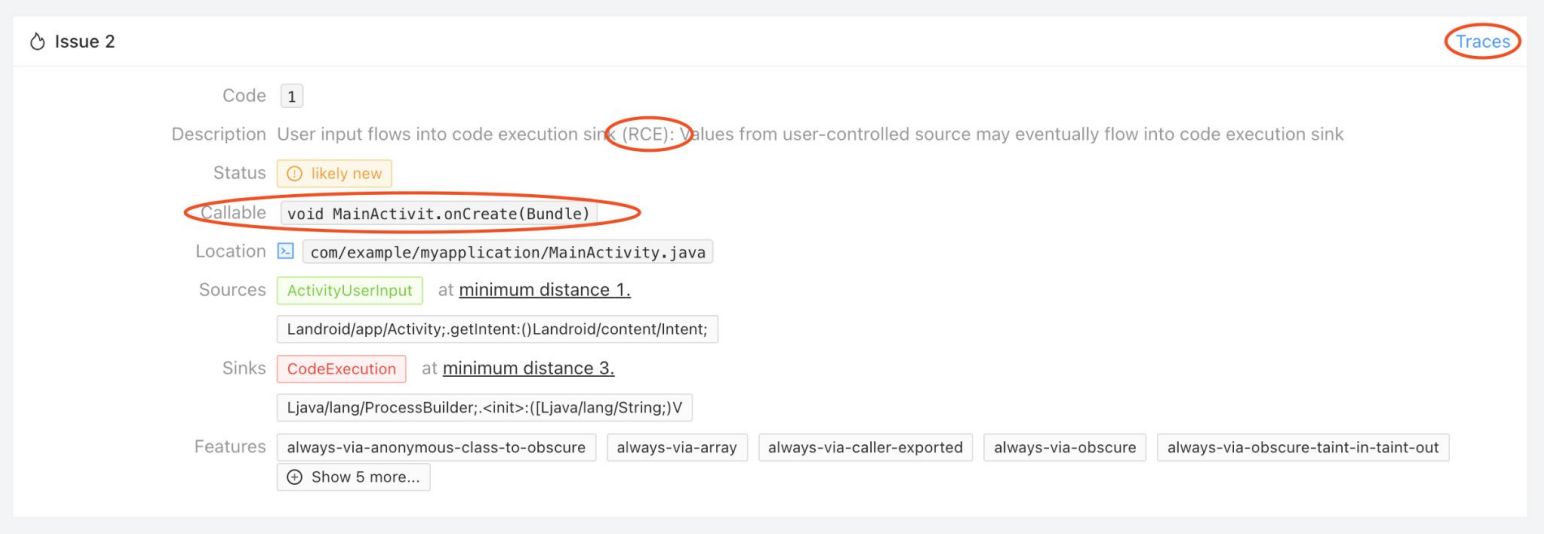
```
@@ -189,0 +189,2 @@
+    <uses-permission-sdk-23
+        android:name="android.permission.USE_FINGERPRINT" />
```

https://github.com/JakeWharton/dependency-tree-diff

Seguridad

Detectar Vulnerabilidades

- Mariana Trench (MT) es una herramienta desarrollada por Meta
- Se utiliza para detectar y prevenir errores de seguridad y privacidad en aplicaciones de Android y Java



https://github.com/facebook/mariana-trench

Muchas Gracias

¿Preguntas?