POLITECNICO MILANO 1863

DIPARTIMENTO DI ELETTRONICA INFORMAZIONE E BIOINGEGNERIA

Lifecycle and Event-Based Testing for Android Applications

Simone Graziussi, Konstantin Rubinov, Luciano Baresi

Problem & Status

Mobile apps are characterized by a great number of **events**, such as lifecycle events, sensor data, connectivity changes, user input, network responses, etc. These events can happen in many **different orders** and **frequencies**

Lifecycle management and **event concurrency** challenges have been largely **overlooked** to date in mobile app testing

Solution

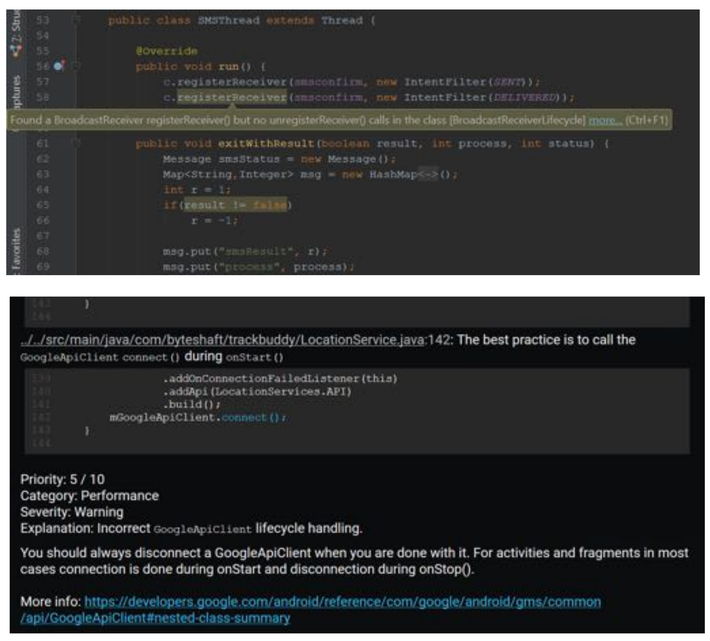
Static analysis to recognize possible misuses of components, and a **dynamic technique** to test app robustness **controlling the critical lifecycle transitions**.

Assertion language to check for race conditions and assess existence, ordering and quantification of the events generated during the application execution

Static and dynamic checking & temporal assertions language

Static analysis:
detect issues in handling of components/resources in accordance to the host Activity/ Fragment lifecycle

Avoid unexpected behaviors or resource waste. Detect early in SW Dev process



Implemented checks:
- Release (e.g., failing to release a resource that was acquired)
- Best practices
- Double instantiation

`BroadcastReceiver, Google API client, Fused Location Provider API, Camera, Ads View`

Dynamic checking with lifecycle test cases:
pre-generated test cases that explore the most common lifecycle changes (e.g., simulate the component being partially hidden)

Developer defines callbacks, while the library manages lifecycle transitions and error reporting

Implemented: pause, stop, rotation, recreation, destruction

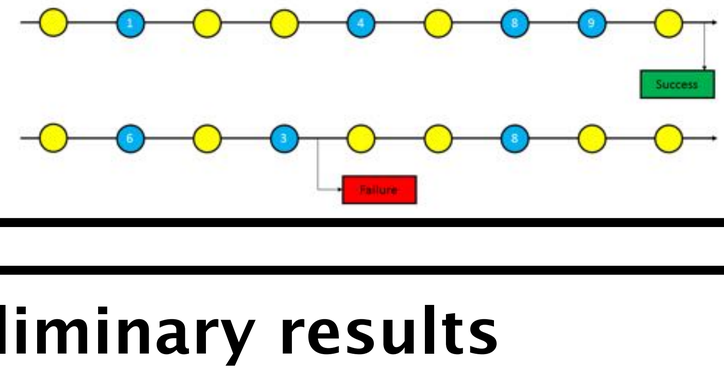
Example rotation test case:

```
public RotationCallback testRotation() {
    return new RotationCallback() {
        private String name;

        @Override
        public void beforeRotation() {
            onView(withId(R.id.first_name_row))
                    .check(matches(isDisplayed()))
                    .perform(click());

            name = "MyFirstName" + (new Random().nextInt(100));
            onView(withId(R.id.my_profile_dialog_input))
                    .check(matches(isDisplayed()))
                    .perform(replaceText(name));

            onView(withText("OK"))
                    .perform(click());

            onView(withId(R.id.first_name))
                    .check(matches(allOf(isDisplayed(), withText(name))));
        }

        @Override
        public void afterRotation() {
            onView(withId(R.id.first_name))
                    .check(matches(allOf(isDisplayed(), withText(name))));
        }
    };}
```

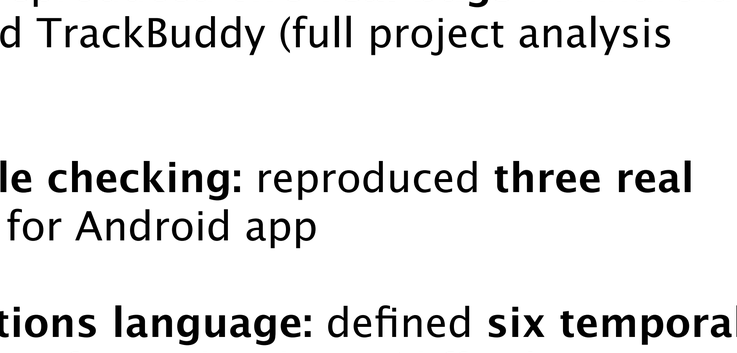
Temporal assertions language:
specify event-related conditions/ consistency checks for standard unit, integration or UI tests

Express `Existence, Order, Causality, Quantification` in concise checkable language; use connectives (logical, implication, etc.) to express complex conditions

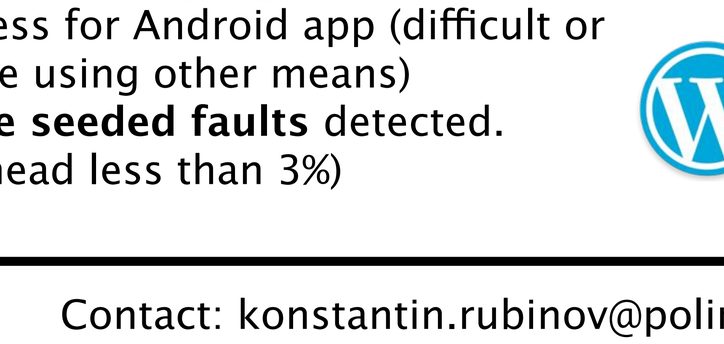
Existence:



Causality:



Order:



Prototype

Static analysis is implemented on top of Android Lint, integrates with Eclipse and Android Studio

Dynamic lifecycle testing library is standalone, works with Espresso and Robolectric

Temporal assertion language is implemented in a library on top of RxJava and RxAndroid

Source code and documentation available at: https://github.com/Simone3/Thesis

Preliminary results

1. Static analysis: reproduced **two real bugs** in Android apps InTheClear and TrackBuddy (full project analysis **20-40 seconds**)

2. Dynamic lifecycle checking: reproduced **three real bugs** in Wordpress for Android app

3. Temporal assertions language: defined **six temporal checks** for Wordpress for Android app (difficult or impossible to define using other means)
Fault seeding: **three seeded faults** detected.
(performance overhead less than 3%)

Contact: konstantin.rubinov@polimi.it