Vincenzo Riccio

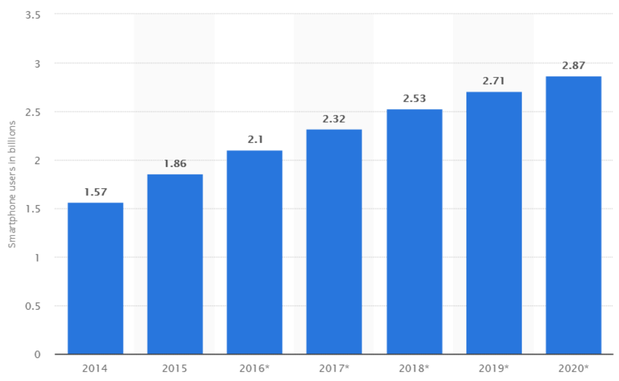
Tutor: Anna Rita Fasolino

XXXI Cycle - II year presentation

A Day in the Lifecycle

Android Activity Lifecycle Testing

Number of smartphone users worldwide from 2014 to 2020 (in billions)

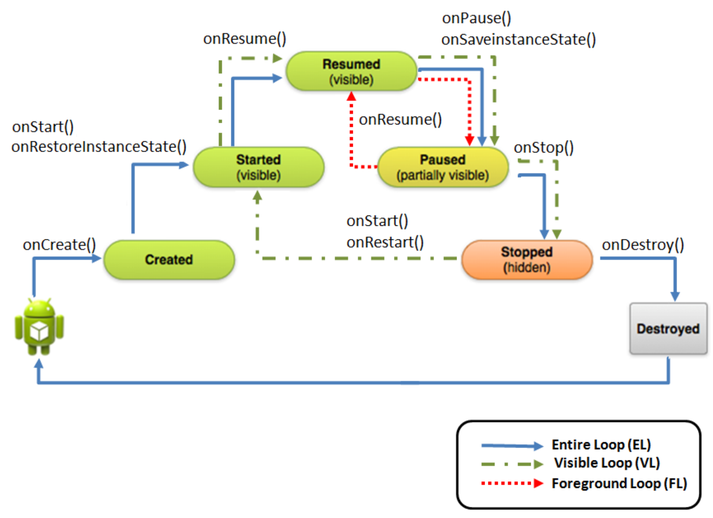


https://www.statista.com/statistics/330695/

- The total number of smartphone users worldwide is forecast to surpass 2.8 billion in 2020
- There is a constant demand for new mobile apps
- The demand for **app quality** has grown together with their spread

Android Activity Lifecycle

- An Android app is composed by one or more Activities. Each Activity represents a single screen
- The Android Framework defines a peculiar lifecycle for Activity instances in order to manage them transparently to the user who can navigate through an app and switch between apps without losing his progress and data



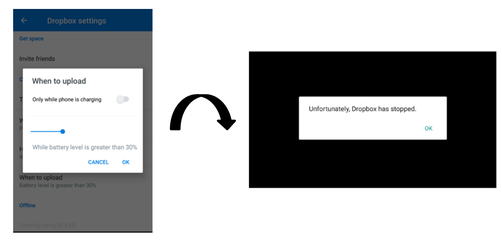
Motivation

Android App developers should correctly implement Activities, taking into account their lifecycle. This ensures the app works the way users expect and does not exhibit aberrant behaviors as it transitions through different lifecycle states at runtime.

Android apps suffer from issues that can be attributed to Activity lifecycle mishandling and affect their quality, such as:

- **Crashes**
- **Graphical User Interface (GUI) failures**
- **Memory Leaks**
- **Threading Issues**

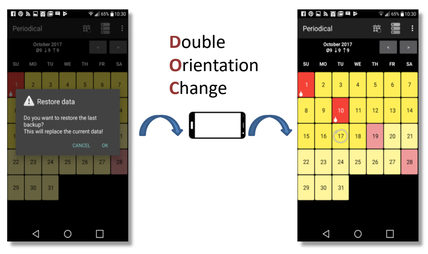
There is the need for specific testing techniques targeting Activity lifecycle conformance [1]



Example of an issue occurring in Dropbox v27.1.2
The orientation change event exercises the Entire Loop (EL) and triggers a crash

Why does the orientation change mess up my Android app?

Exploratory study that investigates the GUI failures exposed in Android apps by changing the screen orientation, a mobile-specific event able to exercise the Activity lifecycle [2]



Double Orientation Change

DOC GUI Failure!!!

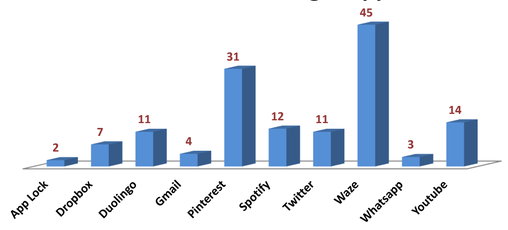
Missing Dialog

code fault & fix

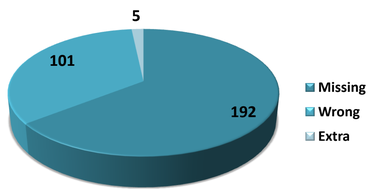
```
private void doBackup(){
    ...
-       final AlertDialog.Builder builder = new AlertDialog.Builder(this);
-       // The Builder class is used for convenient dialog construction...
-       builder.show()
+       DialogFragment backupAlert = new doBackupDialogFragment();
+       backupAlert.show(getSupportFragmentManager(), "backup");
}
...
+ public class BackupDialogFragment extends DialogFragment {
+       @Override
+       public Dialog onCreateDialog(Bundle savedInstanceState){
+         AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
+         // The Builder class is used for convenient dialog construction...
+         return builder.create(); } }
```

`Show()` method called on `Dialog Builder`

DOC GUI failures found in industrial-strength apps



DOC GUI failures found in 68 open-source apps

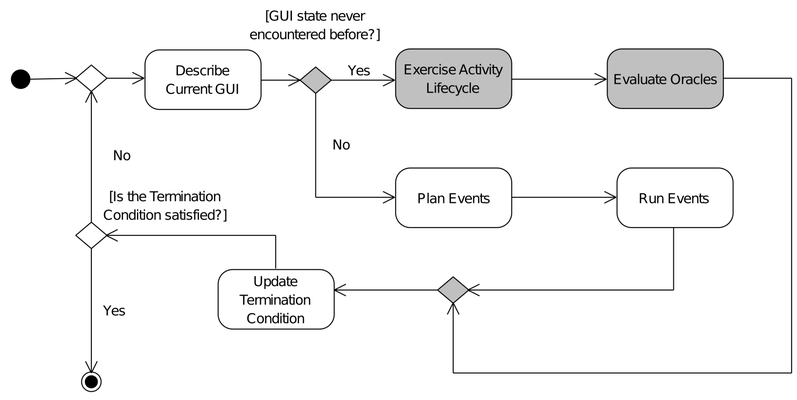


Results

1. 88% of the considered apps are affected by GUI failures due to orientation changes
2. Most of the detected failures involve Dialog objects missing from the GUI after the DOC
3. 6 classes of common faults causing GUI failures have been identified

ALARic (Activity Lifecycle Android Ripper): a Fully Automated Black-box Testing Technique for Android Activities

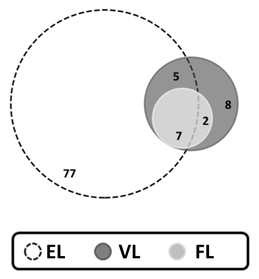
Technique and tool for detecting issues in Android apps that are tied to the Activity lifecycle. It is able to automatically explore the app under test, to systematically exercise the lifecycle of its Activities, and to detect both GUI failures and crashes [3]. It has been defined by extending the generic online GUI testing algorithm described by the framework proposed in [4]



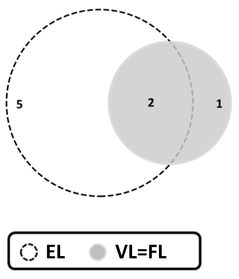
Empirical Evaluation on 15 real Android apps

1. ALARic has been effective in detecting issues tied to the Activity lifecycle in all the analyzed apps
2. ALARic outperformed Monkey, the state-of-the-practice tool, in detecting issues tied to the Activity lifecycle

Detected GUI Failures



Detected Crashes



Contacts

Vincenzo Riccio
Department of Information Technology and Electrical Engineering
Università degli Studi di Napoli Federico II
tel: 081 7683819
email: vincenzo.riccio@unina.it

Collaborations

U.PORTO FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

Research Group

REVERSE — DIETI UNINA

Future Work

- Definition of fault localization techniques aimed at detecting source code bugs that may cause failures tied to the Activity Lifecycle
- Design of novel mutation operators for testing Android apps by exploiting the identified Android-specific fault classes
- Extension of the ALARic tool by implementing a set of oracles able to detect other issues tied to the Activity lifecycle, such as memory leaks and threading issues
- Extension of the proposed approaches to test the lifecycle of other Android app components, such as Services, Fragments and Content Providers

References

[1] Zein S., Salleh N., Grundy J. A Systematic Mapping Study of Mobile Application Testing Techniques. *J. Syst. Softw*. Jul 2016; 117(C):334–356. https://doi.org/10.1016/j.jss.2016.03.065

[2] Amalfitano D., Riccio V., Paiva ACR., Fasolino AR. Why does the orientation change mess up my Android application? From GUI failures to code faults. *Softw Test Verif Reliab Journal* , Wiley, 2017. https://doi.org/10.1002/stvr.1654

[3] Amalfitano D., Amatucci N., De Simone V., Riccio V., Fasolino AR. Is This the Lifecycle We Really Want? An Automated Black-Box Testing Approach for Android Activities. *Submitted to 11th IEEE Conference on Software Testing, Validation and Verification (ICST) 2018*

[4] Amalfitano D, Amatucci N, Memon AM, Tramontana P, Fasolino AR. A general framework for comparing automatic testing techniques of Android mobile apps. *Journal of Systems and Software* 2017; 125:322–343. https://doi.org/10.1016/j.jss.2016.12.017