4th International Workshop on TESTing Techniques & Experimentation Benchmarks for Event-Driven Software

Luxembourg
March 19, 2013

Considering Context Events in Event-Based Testing of Mobile Applications

University of Naples "Federico II", Italy

DIPARTIMENTO DI INGEGNERIA ELETTRICA E TECNOLOGIE DELL'INFORMAZIONE

- Domenico Amalfitano
- Anna Rita Fasolino
- Porfirio Tramontana
- Amatucci Nicola

DIETI

Context and Motivations

- Context:
  - Testing of Android Mobile Applications.
- Motivations:
  - Testing the Android Apps by taking into account their context and context-related events.

Mobile Applications as EDS Systems

- Mobile apps are event-driven systems.
  - They are able to sense and react to a wide set of events besides user ones.

- Mobile devices are equipped with a wide variety of hardware sensors that are able to sense the *context in which the device stays and to notify context changes to the running app by means of events*.

- The user can be considered as a part of the context of an app

Types of mobile context events

- Types of context event are:
  - user events produced through the GUI;
  - events coming from the external environment and sensed by device sensors (such as temperature, pressure, GPS, geomagnetic field sensor, etc.);
  - events generated by the device hardware platform (such as battery and other external peripheral port, like USB, headphone, network receiver/sender, etc.);
  - events typical of mobile phones (such as the arrival of a phone call or a SMS message);
  - events like the arrival of an e-mail or social networks notifications, that are related to the fact that modern mobile phones are more and more Internet connected.

MobileApps and Apps4Mobile

- Not all the mobile apps are designed to react to context-related events not coming from the GUI.
- **MobileApps**, react to all contextual events (both GUI and non-GUI ones).
- **App4Mobile**, react only to GUI events
  - they are just traditional applications that have been rewritten to run on mobile devices.

H. Muccini, A. Di Francesco, P. Esposito. Software Testing of Mobile Applications: Challenges and Future Research Directions. Proceedings of the 7th International Workshop on Automation of Software Test (AST), 2012,

How to test the MobileApps

- The category of App4Mobile may be effectively tested by means of testing techniques applicable to traditional applications.
- MobileApps requires to be tested by event-based testing techniques that properly consider all types of context-related events.
  - This may be very expensive due to the large number of possible contexts, event classes and combinations of events and contexts to be considered.
- Effective strategies of testing for MobileApps are needed.

Some possible testing strategies

- The application behaviour has to be checked in response to several types of context event.
- Effective strategies for test case generation should be used to define sequences of events of mixed types.
- Simple Strategies
- Systematic Strategies

“Simple” strategies

- May define event sequences trying to achieve the coverage of each class of contextual events with a fair policy.
- Not requiring any specific knowledge about the app under test.
- Help to discover crashes or freezes, often reported in bug reports, that may occur when an app is impulsively solicited by contextual events like:
  - Notification of a connection/disconnection of a plug (USB, headphone, …)
  - Incoming of a phone call,
  - Notification of the GPS signal loss (for instance when the device enters a tunnel).

“Systematic” strategies

- May require the coverage of specific event sequences representing specific *usage scenarios of the application*.
- Are scenario-based strategies that:
  - define relevant ways of exercising an application by firing sequences of events.
  - may be described by different formalisms.
- In scenario-based testing, suitable techniques for defining scenarios are needed.
  - an interesting approach may be based on “event-patterns”

Event - Patterns

- Representations of peculiar event sequences that abstract meaningful test scenarios.
- can be defined as a notable sequence of contextual events that may be used to exercise the application.
- Possibly trigger a faulty behaviour of the application.
- Are often used for rapid testing of embedded systems

Event – Patterns Specification

- Each Event – Pattern may be specified by:
  - a name,
  - a textual description,
  - the corresponding event sequence that must include one or more events.
    - defined by appropriate regular expressions.

- For automating test execution, each event-pattern can be associated with a test class that exposes an **`execute`** method able to trigger the pattern's sequence of events.

- An event-pattern may be included in other event sequences, or used in isolation to test an app.

- An event-patterns repository have been manually populated after a preliminary analysis conducted on the bug reports of open source applications.

Event Pattern Example

Event-Pattern Name	Event-Pattern Description	Event-Pattern Specification
LRGPS	Loss and successive Recovery of GPS signal while walking	(locationChange)+, GPSLoss, GPSRecovered, (locationChange)+
NI	Network Instability	(NetworkEnabled,NetworkDisabled)+
EGSW	The user Enables the GPS provider through the Settings menu and starts a Walk	openSettings, GPSOn, (locationChange)+
SIE	Arrival of a phone call when the device is in Stand-by. Then the call ends before the user accepts it.	standBy, IncomingPhoneCall, phoneCallEnds
UP	USB plugging in after any other event except the event itself	^USBPlugged,USBPlugged
MSVC	Magnetic Sensor value changed after any other event except the event itself.	^magneticSensor ValueChange, magneticSensor ValueChange
IPC	Incoming of a phone call after any other event except the event itself.	^IncomingPhoneCall, IncomingPhoneCall

Event Pattern Example

Event-Pattern	Event-Pattern Description	Event-Pattern Specification

Pattern	Pattern Class Signature	Execute Public Method

EGSW	public Class EGWS {
	...
	public void execute(Point A, Point B, ArrayList<Point> Route);
	private void OpenSettings();
	private void EnableGPS();
	private void Navigate(Point A, Point B, ArrayList<Point> Route);
	...
	}
		public void execute (Point A, Point B, ArrayList <Point> Route){
		OpenSettings();
		EnableGPS();
		Navigate(A,B,route);
		}

LRGPS	public Class LRGPS {
	...
	public void execute(Point A, Point Y, Point B, ArrayList<Point> Route);
	private void GPSLocationChange(Point X);
	private void GPSLoss();
	private void GPSRRecovered();
	private void Navigate(Point A, Point B, ArrayList<Point> Route);
	...
	}
		public void execute (Point X, Point Y, Point B, ArrayList<Point> Route) {
		GPSLocationChange(X);
		GPSLoss();
		GPSRRecovered();
		GPSLocationChange(Y);
		Navigate(Y+1,B,Route);
		}

SIE	public Class SIE {
	...
	public void execute();
	private void deviceGoesInStandBy();
	private void incomingPhoneCall();
	private void phoneCallEnds();
	...
	}
		public void execute() {
		deviceGoesInStandBy();
		incomingPhoneCall();
		phoneCallEnds();
		}

the event itself.

Approaches for using event-patterns for testing mobile applications

- Test cases can be generated by exploiting the event – patterns stored in the repository.
- We show three examples of testing techniques that exploit the event-patterns for testing a mobile app.
  - T1: Manual technique
  - T2: Mutation-based technique
  - T3: Exploration-based technique

T1 - Manual technique

- Scenario-based test cases that include one or more instances of event-patterns are manually generated.
- The tester can add the needed assertions manually, or test cases can just check the occurrence of crashes.

Example of using T1

- Suppose that the tester wants to test the app behaviour in the following scenario:

> *The user activates the GPS provider in the settings menu of the application, and begins to cross the path that goes from point A to point B through N points. At point X of the navigation, the application loses the GPS signal and recovers it at the point Y.*

- This scenario includes instances of two event-patterns EGSW and LRGPS.

Example of using T1

- Suppose that the tester wants to test the app behaviour in the following scenario:

> *The user activates the GPS provider in the settings menu of the application, and begins to cross the path that goes from point A to point B through N points. At point X of the navigation, the application loses the GPS signal and recovers it at the point Y.*

- This sce
patterns

```
public void testScenario00038() {
        EGSW.execute(A,X-1,routeAX);
        LRGPS.execute(X,Y,B,routeYB);
}
```

T2 - Mutation-based technique

- Event-patterns are used to modify existing test cases, by applying mutation techniques.

- Event-pattern sequences are placed inside already existing test cases that are defined:
  - manually
  - by Capture techniques
  - automatically (I.e. by a Ripper).

Example of using T2

- The absence of crashes in an application scenario where, after any user event, the device goes in stand-by and a phone call comes (pattern SIE) must be tested.

Test Case before mutation	Mutated test case

`public void testTrace00004() {`
 	
`fireEvent (16908315, 16, "OK", "button", "click");`
 	
`fireEvent (2131099651, 6, "", "button", "click");`
 	
`fireEvent (0, "", "null", "openMenu");`
`}`
	`public void testTrace00004_EP_SIE() {`
	
	`fireEvent (16908315, 16, "OK", "button", "click");`
	
	`SIE.execute();`
	
	`fireEvent (2131099651, 6, "", "button", "click");`
	
	`SIE.execute();`
	
	`fireEvent (0, "", "null", "openMenu");`
	
	`SIE.execute();`
	`}`

T3 - Exploration-based technique

- Event-patterns are used in automatic black-box testing processes based on dynamic analysis of mobile apps.

```
StartApplication();
        while (! termination Criterion) {
                ExtractCurrentSensingEvents();
                PlanTasks();
                RunNextTask();
        }
}
```

Assessing the feasibility of the proposed techniques

- The feasibility of the proposed techniques was preliminary assessed in the context of Android mobile applications.
  - the proposed techniques are applicable to any mobile technology.
- Two main technological problems related to the Android platform were solved:
  - defining a solution for dynamically recognizing the context event classes which the app is able to sense and react at a given time (i.e., implementing the operation `ExtractCurrentSensingEvents` in T3);
  - defining techniques for triggering the context events (i.e., implementing the `RunNextTask` operation in T3).

A first exploratory case study

- To asses how the effectiveness of an event-based testing technique varies when context events, not only GUI events, are taken into account.
- The proposed Exploration-based technique T3 was analyzed.

Experiment Setup

- Five real-world MobileApp Android applications were considered.
- Each of them was tested by the T3 technique twice.
  - The first time, simple patterns made by only user events were exercised.
  - The second time, simple patterns including a different type of context event were executed.
- The effectiveness, in terms of code coverage, achieved by the two executions was compared

Extended Android Ripper

- The Android Ripper tool was exploited to perform the experiment.
  - http://wpage.unina.it/ptramont/GUIRipperWiki.htm
- First the Android Ripper was configured to trigger only user events
- Lastly the Extended Ripper version was configured to fire context events such as:
  - location changes, enabling/disabling of GPS, changes in orientation, acceleration changes, reception of SMS messages and phone calls, shooting of photos with the camera.
- Both versions of the Ripper are able to systematically explore the app under test by searching for crashes and to measure the obtained code coverage.

Results

App	LOC Coverage		Method Coverage	
	Android Ripper	Extended Ripper	Android Ripper	Extended Ripper
Marine Compass	435 (92%)	463 (97%)	25 (86%)	27 (93%)
Bubble Level	371 (60%)	464 (75%)	75 (65%)	85 (74%)
Pedometer	528 (66%)	544 (67%)	160 (71%)	161 (72%)
Omnidroid	3409 (56%)	3480 (57%)	789 (58%)	813 (60%)
Wordpress	4505 (45%)	4599 (46%)	779 (53%)	784 (53%)

Results

App	LOC Coverage		Method Coverage	
	Android Ripper	Extended Ripper	Android Ripper	Extended Ripper
Pedometer	528 (66%)	544 (67%)	160 (71%)	161 (72%)
Omnidroid	3409 (56%)	3480 (57%)	789 (58%)	813 (60%)
Wordpress	4505 (45%)	4599 (46%)	779 (53%)	784 (53%)

Methods and LOC that implement the Sensor's Handlers were covered

Results

App	LOC Coverage		Method Coverage	
	Android Ripper	Extended Ripper	Android Ripper	Extended Ripper
Marine Compass	435 (92%)	463 (97%)	25 (86%)	27 (93%)
Bubble Level	371 (60%)	464 (75%)	75 (65%)	85 (74%)
Omnidroid	3409 (56%)	3480 (57%)	789 (58%)	813 (60%)
Wordpress	4505 (45%)	4599 (46%)	779 (53%)	784 (53%)

The source code has just one context-related event handler.
It is responsible for managing the acceleration change notification event.

Results

App	LOC Coverage		Method Coverage	
	Android Ripper	Extended Ripper	Android Ripper	Extended Ripper
Marine Compass	435 (92%)	463 (97%)	25 (86%)	27 (93%)
Bubble Level	371 (60%)	464 (75%)	75 (65%)	85 (74%)
Pedometer	528 (66%)	544 (67%)	160 (71%)	161 (72%)
Wordpress	4505 (45%)	4599 (46%)	779 (53%)	784 (53%)

Event handlers related to the management of incoming phone calls and SMS messages were covered

Results

App	LOC Coverage		Method Coverage	
	Android Ripper	Extended Ripper	Android Ripper	Extended Ripper
Marine Compass	435 (92%)	463 (97%)	25 (86%)	27 (93%)
Bubble Level	371 (60%)	464 (75%)	75 (65%)	85 (74%)
Pedometer	528 (66%)	544 (67%)	160 (71%)	161 (72%)
Omnidroid	3409 (56%)	3480 (57%)	789 (58%)	813 (60%)

camera photo shoot notification event handler, and the location change notification handler were covered.

Conclusion

- Three proposals of techniques for event based testing of mobile applications have been presented.
- A testing technique that represents an extension of the Android Ripping technique has been presented.
- Some preliminary case studies conducted on real world Android apps demonstrate the effectiveness of the implemented technique.

Future Works

- Address wider experimentation in order to assess the effectiveness of the proposed techniques.
- Build an event-patterns repository by analysing a large corpus of bug reports related to mobile apps
- Consider events causing the interaction between different components of the same application or different applications, by adding to the Extended Ripper new features supporting Intent Messages generation and execution.

Thank you for your attention !!!