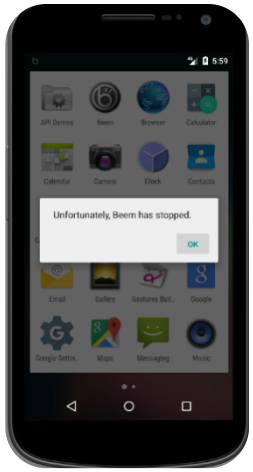
Repairing crashes in Android Apps
安卓应用的程序崩溃修复

Shin Hwei Tan* Zhen Dong^ Xiang Gao^ Abhik Roychoudhury^

Southern University of Science and Technology*
National University of Singapore^



Shin Hwei Tan 陈馨慧

Website: https://www.shinhwei.com/

Email: tansh3@sustc.edu.cn

- 程序自动修复
- 测试用例修复
- 代码注释分析
- 软件测试
- 基于搜索的软件工程

伊利诺伊大学厄巴纳-香槟分校硕士

伊利诺伊大学厄巴纳-香槟分校学士

UIUC

新加坡国立大学博士

南方科技大学助理教授

写bug了！

ICSE 40TH INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING MAY 27 - JUNE 3 2018 GOTHENBURG, SWEDEN

Attending Program Tracks Committees Search Series

* ICSE 2018 * (series) / Technical Papers /

Repairing Crashes in Android Apps

Who	*Shin Hwei Tan, Zhen Dong, Xiang Gao, Abhik Roychoudhury*
Track	* ICSE 2018 * Technical Papers
When	**Wed 30 May 2018 15:00 - 15:20 at H2 room** - Apps and App Stores II Chair(s): Patrick Maeder
Link to Preprint	http://www.shinhwei.com/droixicse_camera.pdf

能工具 名为SapFix

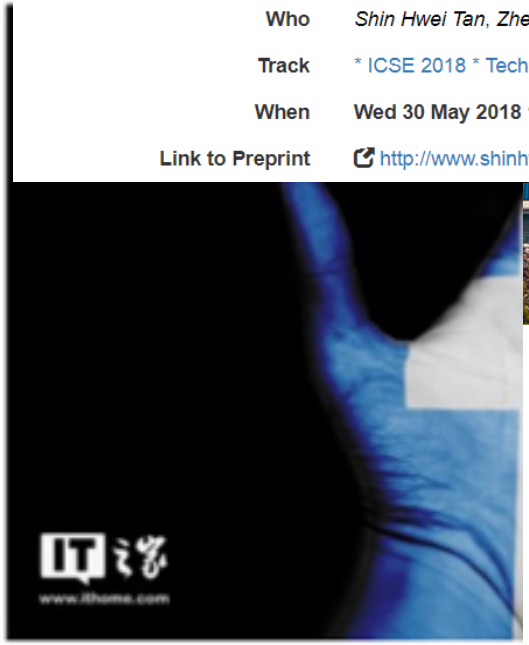
w.zdnsh.com 作者：赤壁点讯网

)工具，可以自动扫描代码，寻找漏洞，然后测试

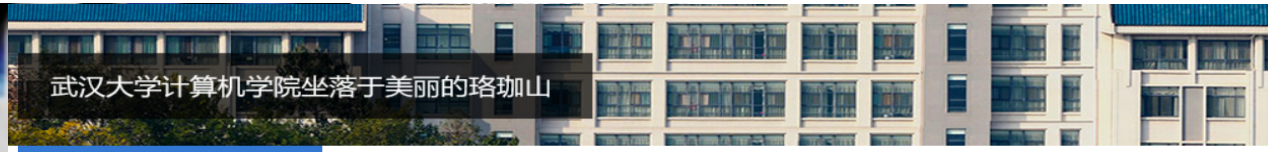
了一种名为SapFix的人工智能(AI)工具，可以自动

荐最佳修补方案。

目前已经应用于Facebook庞大的代码库。该公司还



还表示，对于高攻击错误，SapFix能创建补丁，对需求
x会从一组修复模板中进行提取，以创建补丁。当该过
ook表示，并没有视SapFix为人类程序员的替代品，而
更多的人接受编程。
青岛it培训机构，it培训，it培训价格，it培训哪家好，



通知动态
notification

学院通知
公示公告
新闻动态
学术讲座

学术讲座

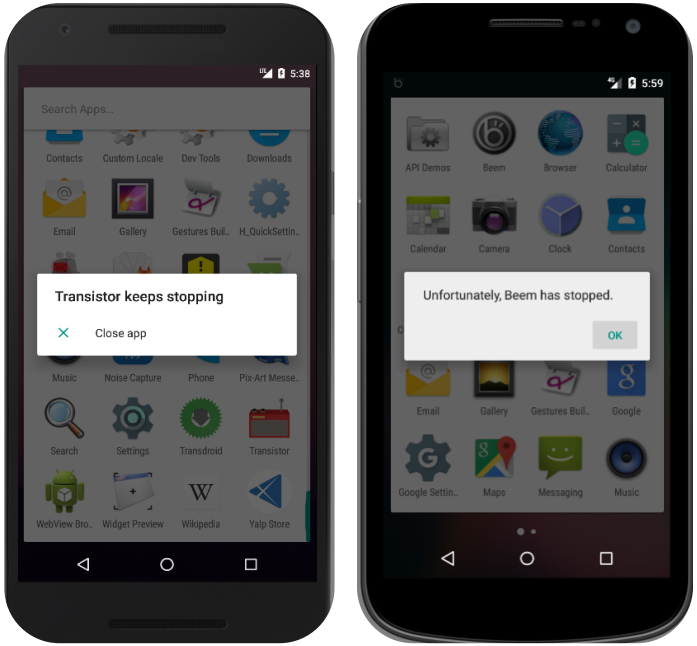
6月22日学术报告（陈馨慧 南方科技大学）

2018年06月20日09时 0人评论 64次

报告题目： 安卓应用的程序崩溃修复 Repairing crashes in Android apps
报告时间： 6月22日10:30 – 11:20
报告地点： 计算机学院E202
报告人： 陈馨慧
报告人国籍：马来西亚
报告人单位：南方科技大学

报告人简介：
陈馨慧，女，南方科技大学计算机科学与工程系助理教授。2010和2012年获伊利诺伊大学厄巴纳-香槟分校（U
业于新加坡国立大学（NUS），获计算机博士学位。研究方向包括自动程序修复，软件测试与软件分析等多个方
发表学术论文10余篇。曾荣获2015年Google Anita Borg Memorial Scholarship。

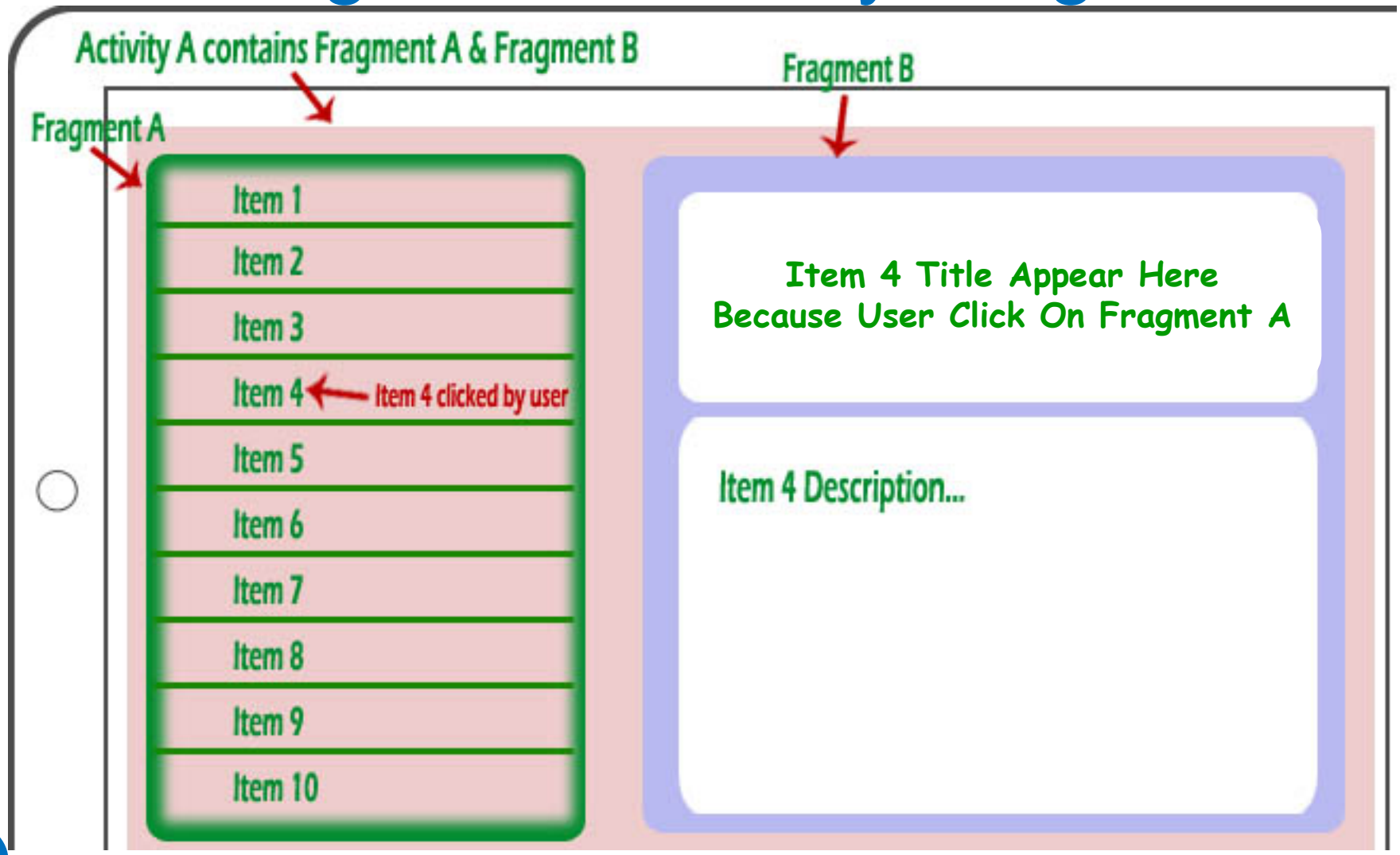
Android Repair System



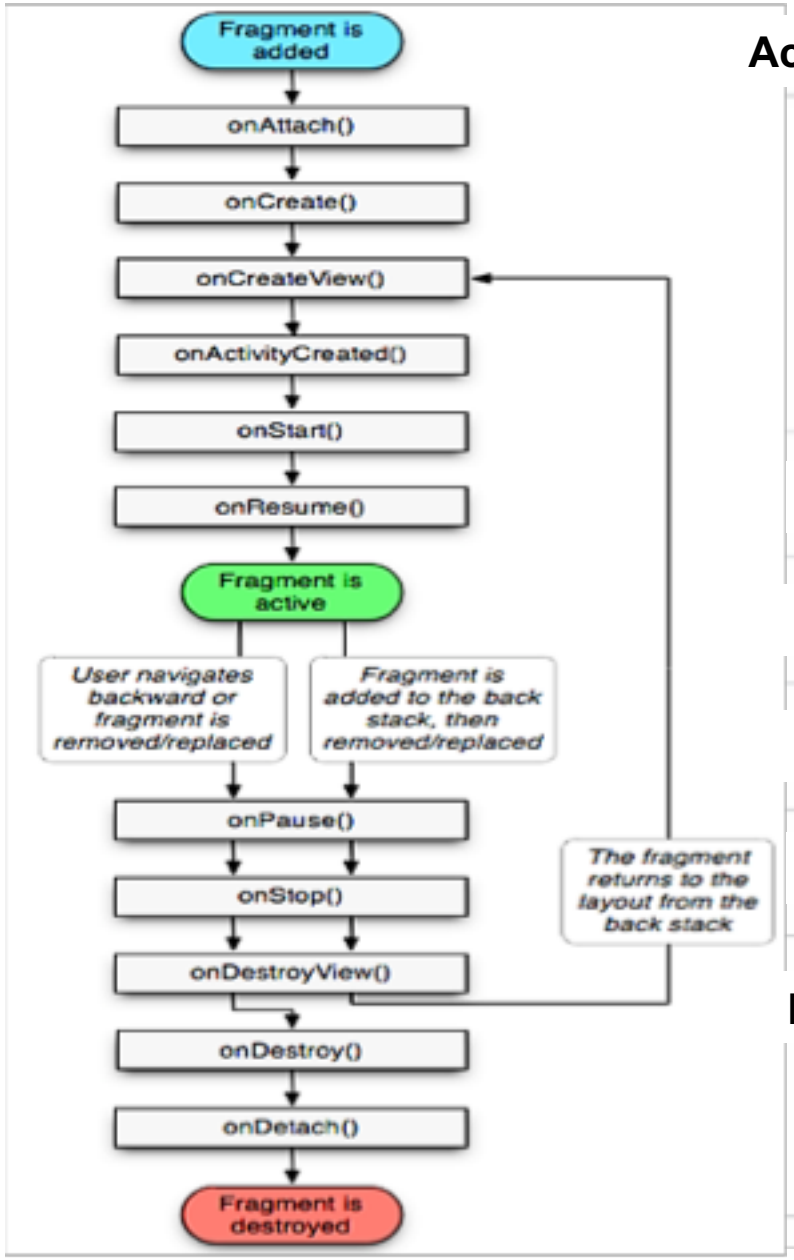
Criteria for Android repair system:

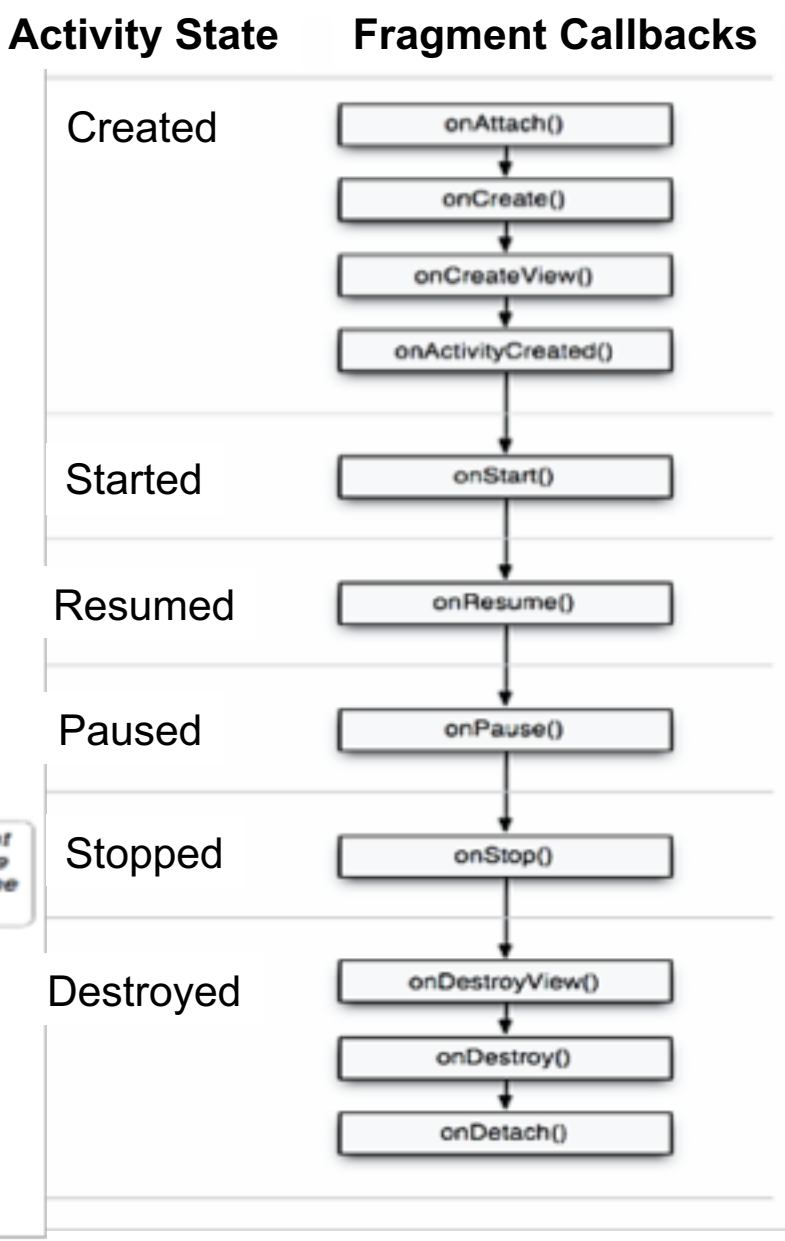
- ✓ Could be used by any smartphone user
  - ✓ Doesn't require source code
  - ✓ No prior knowledge of Android app
  - ✓ No prior experience of running tests

Background: Activity/Fragment



Complexity of Android Activity/Fragment Lifecycle





What are the common causes of crashes in Android apps?

- Obtain popular Android apps written in Java from 

- For each app, search for closed bug reports
  - >7500 bug reports
- Filter crash-related bug reports
  - 1155 bug reports
  - ➢ 15% of bug reports in Android apps are crash-related
- Filter bug reports with bug-fixing commits
  - 107 GitHub Issues from 15 apps

What lead to crashes in Android apps?

Category	Specific reason	Description	GitHub Issues (%)	Frequent Exception Type	Category Total (%)
Lifecycle	Configuration changes	activity recreation during configuration changes	5.61	NullPointer	14.02
	Stateloss	transaction loss during commit	2.80	IllegalState	
	GetActivity	activity-fragment coordination	2.80	IllegalState	
	Activity backstack	inapproapriate handling of activity stack	1.87	IllegalArgument	
	Save instance	uninitialized object instances in onSaveInstance() callback	0.93	IllegalState	
Resource	Resource-related	resource type mismatches	10.28	NullPointer	16.82
	Resource limit	limited resources	4.67	OutOfMemory	
	Incorrect resource	retrieve a wrong resource id	1.87	SQLite	
Callback	Activity-related	missing activities	7.48	NullPointer	17.76
	View-related	missing views	6.54	NullPointer	
	Intent-related	missing intents	3.74	NullPointer	
	Unhandled callbacks	missing callbacks	2.80	NullPointer	
Others	Missing Null-check	missing check for null object reference	12.15	NullPointer	52.34
	External Service/Library	defects in external service/library	8.41	NullPointer	
	Workaround	temporary fixes for defect	4.67	IndexOutOfBound	
	API changes	API version changes	2.80	SQLite	
	Others	project-specific defects	24.30	-	

Common root causes of crashes in apps:

- Lifecycle-related (14.02%)
- Resource-related (16.82%)

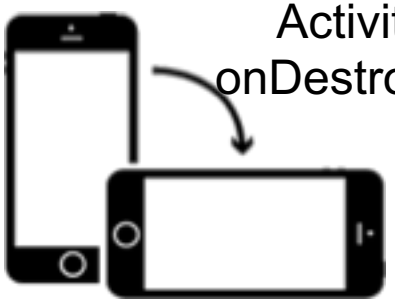
Could be automatically repaired

- Workaround: 4.67 %
- Most common exceptions
  - NullPointerException (40.19%)
  - IllegalStateException (7.48%)

Lifecycle-aware Repair Operators

Operator	Description
S1: GetActivity-check	Insert a condition to check whether the activity containing the fragment has been created.
S2: Retain object	Store object and load it when configuration changes
S3: Replace resource id	Replace resource id with another id of same type.
S4: Replace method	Replace the current method call with another method call with similar name and compatible parameter types.
S5: Replace cast	Replace the current type cast with another compatible type.
S6: Move stmt	Removes a statement and add it to another location.
S7: Null-check	Insert condition to check if a given object is null.
S8: Try-catch	Insert try-catch blocks for the given exception.

Lifecycle-aware Repair Operator: Retain stateful object



Activity re-created:
onDestroy()→onCreate()

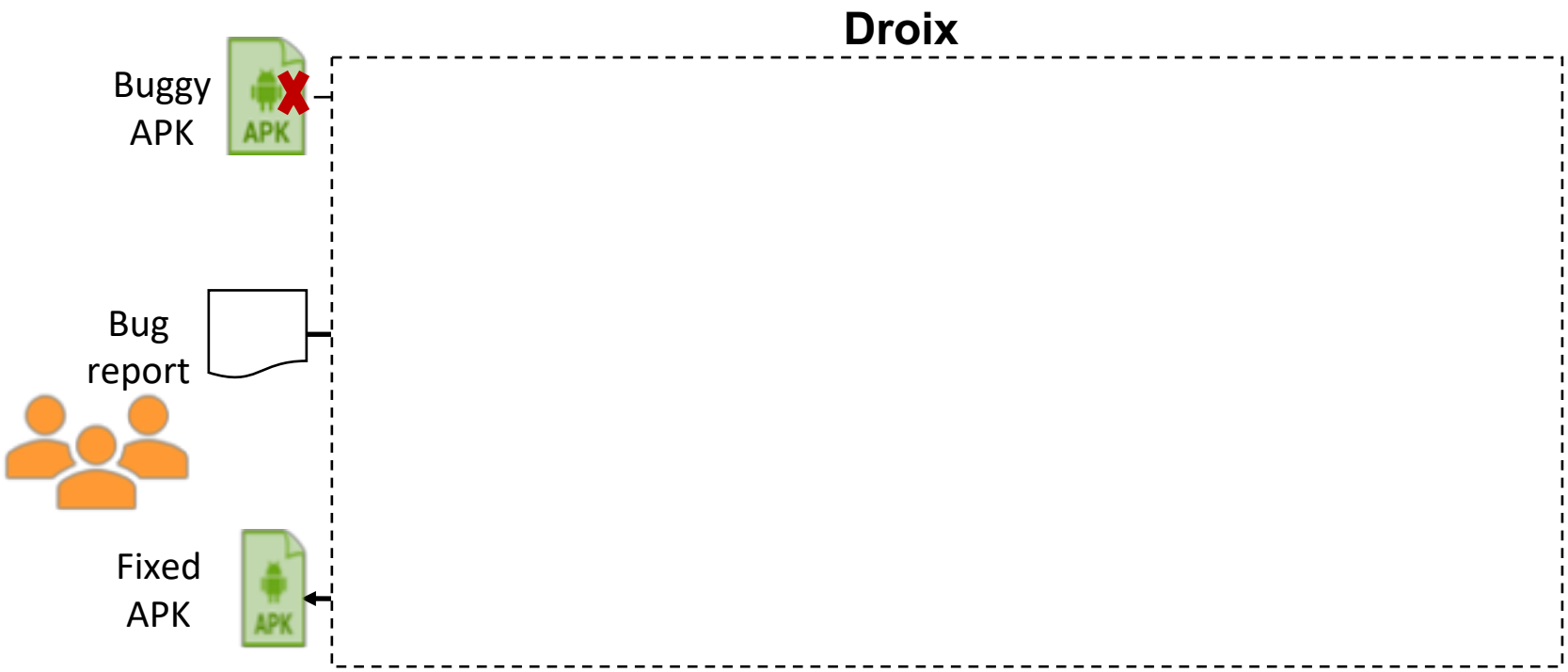
```
public void onCreate(Bundle savedInstanceState) {…
+   setRetainInstance(true);// retain this fragment
}

// new field for saving the object
+ private static Option saveOption;

public View onCreateView(LayoutInflater inflater,
    ViewGroup container, Bundle savedInstanceState) {
// saving and loading the object
+   if(option!=null){
+      saveOption = option;
+   }else{
+      option = saveOption;
+   }
✖    switch (option.getButtonStyle()){
```

1

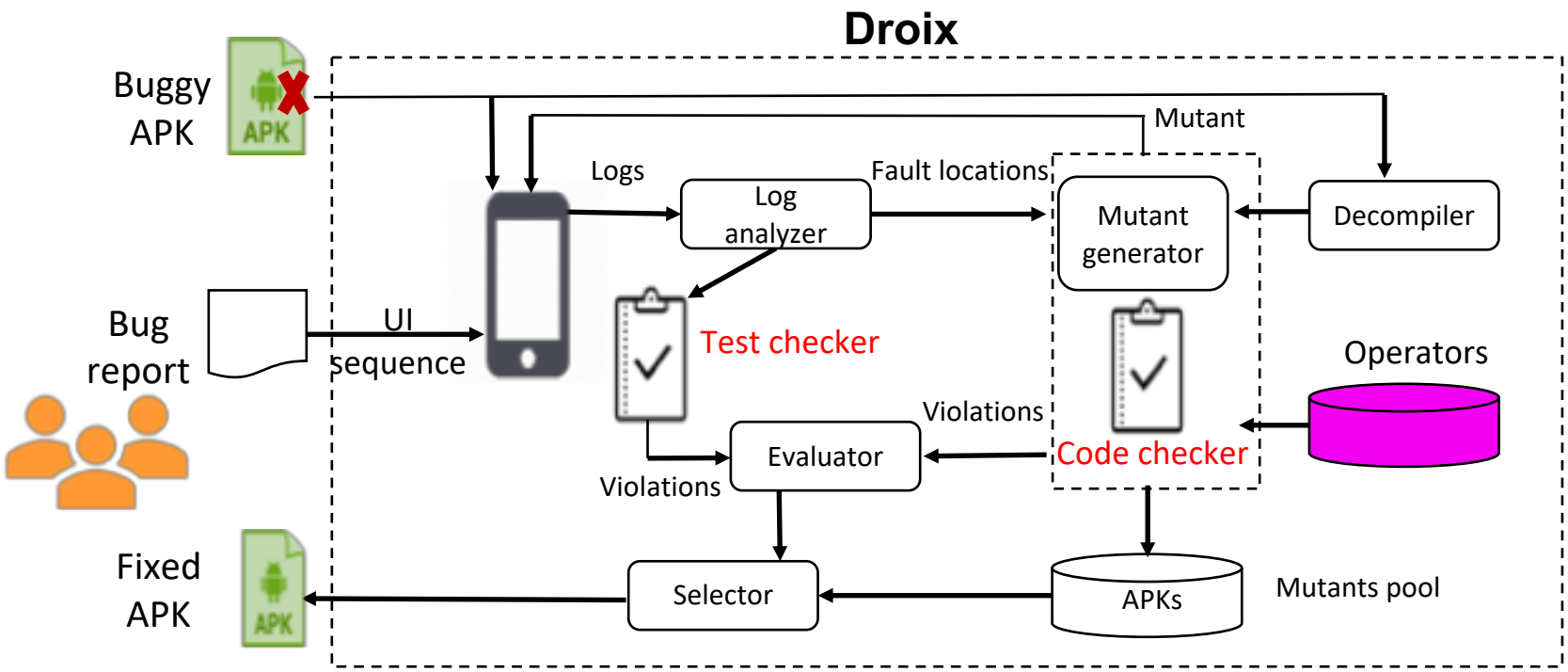
2

3

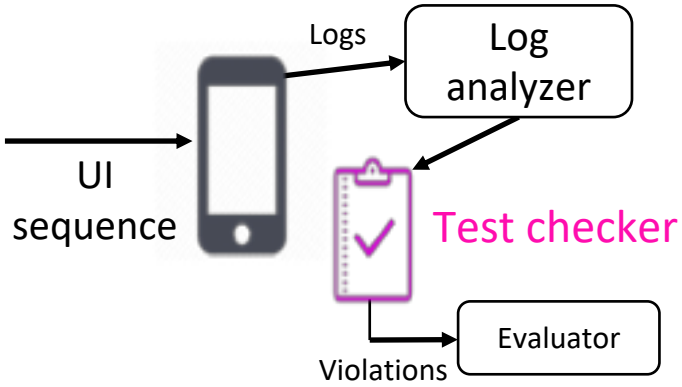
Android Apps Repair System (Droix)



Android Apps Repair System (Droix)

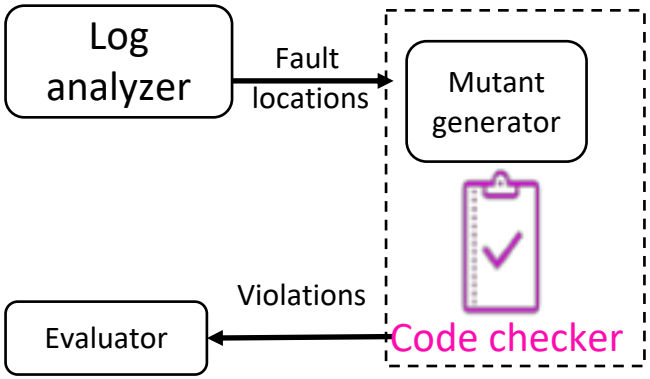


Test-Level Properties Checking



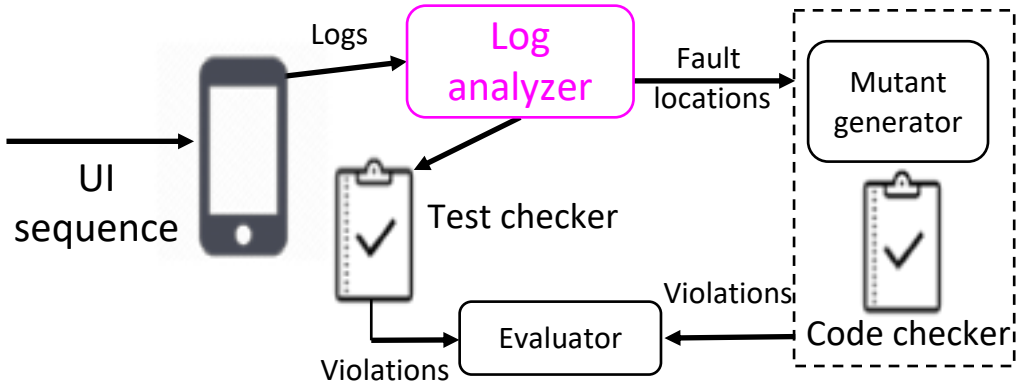
1. Record activity transitions in logs during test replay
2. Given activity transitions *current-state*→*next-state*, check for violations to Android activity/fragment lifecycle management rules

Code-level Properties Checking



For each program transformation by the mutant generator, check if it is valid in terms of compilation errors and coding conventions.

Locating the fault



- ✓ Do not required source code & heavy test suite
  - Log analyzer extracts stack trace information:
    - Exception Type
    - Lines in code where exception is thrown
    - Method calls in the stacktrace
  - Log analyzer extracts callbacks invoked during execution of UI sequence

Code-level & Test-level Properties in *Droix*

Level	Type	Description
Code-level	Well-formedness	Compilable and structural type of program matches with the context of selected operator.
	Bug hazard	Transformation violates Java exception-handling best practices.
	Exception Type	Transformation matches given exception type.
Test-level (Android-specific)	Lifecycle	Event transition matches with the activity/fragment lifecycle model.
	Activity-Fragment	Interaction between a fragment and its parent activity matches the activity-fragment coordination model
	Commit	Commit of fragment's transactions in allowed states.

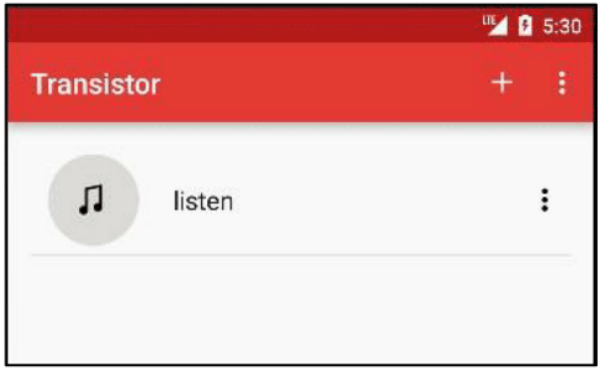
Bug hazard: "What not to do when handling exception?"
❖ Similar concepts as *anti-patterns [FSE'16]*

Searching for Patches

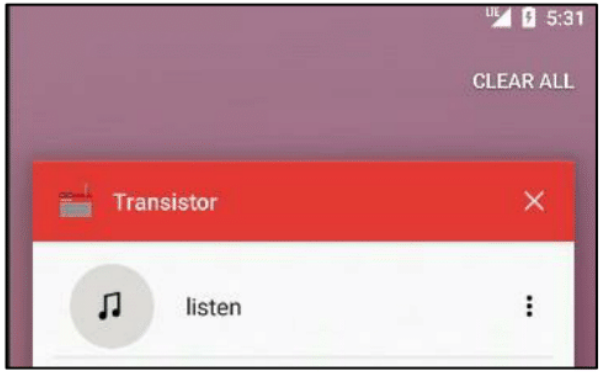
- $(\mu + \lambda)$ Evolutionary Algorithm, μ=40, λ=20
- Code-level & Test-level Properties are divided into:
  - Hard Constraints: eliminate patches if property is violated
    - Well-formedness
    - Bug hazard: inserting try-catch block for runtime errors
  - Soft Constraints: possible property violation
- Fitness Function *f*:

  f = Minimize # of violations for soft constraints

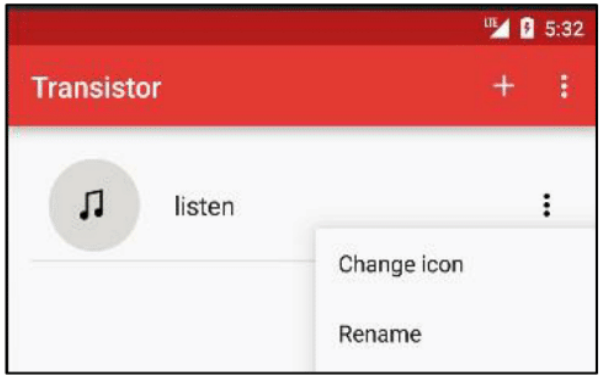
Example: Transistor



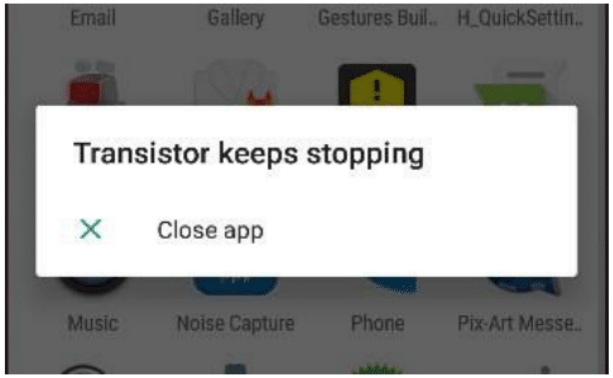
(a) Open Transistor



(b) Press back



(C) Open again and change an icon

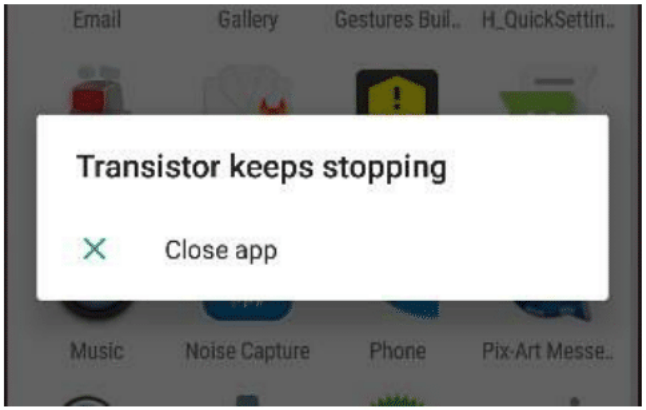


(d) Crashed with a notification

```
FATAL EXCEPTION: MAIN
PROCESS: ORG.Y20K.TRANSISTOR, PID: 2416
JAVA.LANG.ILLEGALSTATEEXCEPTION:
   FRAGMENT MAINACTIVITYFRAGMENT{82E1BEC}
            NOT ATTACHED TO ACTIVITY
```

- Fragment is detached from the previous activity during app termination & should be attached to new activity in the restarting app.

Example: Transistor



(d) Crashed with a notification

Developer's patch

```
- startActivityForResult ( pickImageIntent , REQUEST_LOAD_IMAGE);
+ mActivity.startActivityForResult ( pickImageIntent , REQUEST_LOAD_IMAGE);
```

❖ Reuse old activity (`mActivity`)

Droix's patch

```
+ if ( getActivity() !=null )
    startActivityForResult ( pickImageIntent , REQUEST_LOAD_IMAGE);
```

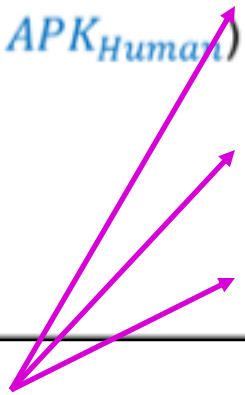
❖ Check if activity containing the fragment has been created.

Which patch is better

?

How to compare two patches?

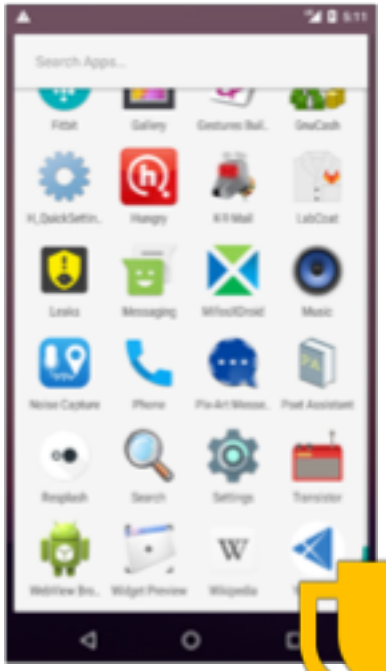
Level	Patch Quality	Meaning
Source-level (Src_{Droix} vs. Src_{Human})	Syntactically Equiv.	Are Src_{Droix} and Src_{Human} exactly the same patch?
	Semantically Equiv.	Do Src_{Droix} and Src_{Human} convey the same meaning?
UI-level (APK_{Droix} vs. APK_{Human})	UI-behavior Equiv.	Do APK_{Droix} and APK_{Human} behaves the same?
	Incorrect	Do APK_{Droix} introduce undesirable behavior?
	Better	Is APK_{Droix} better than APK_{Human} in terms of held-out test?

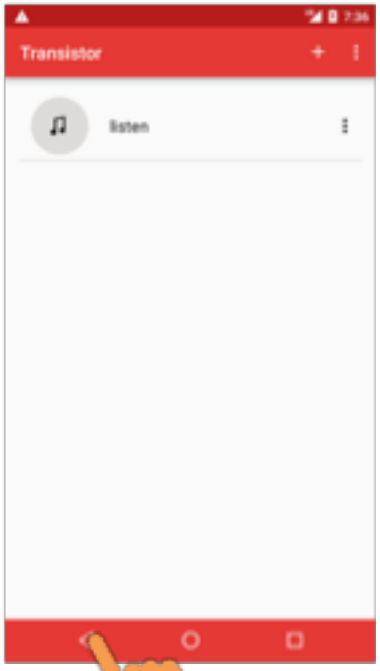


Newly added classification: allow more detailed comparison at the GUI level

Which patch is better?

Patch generated by Droix





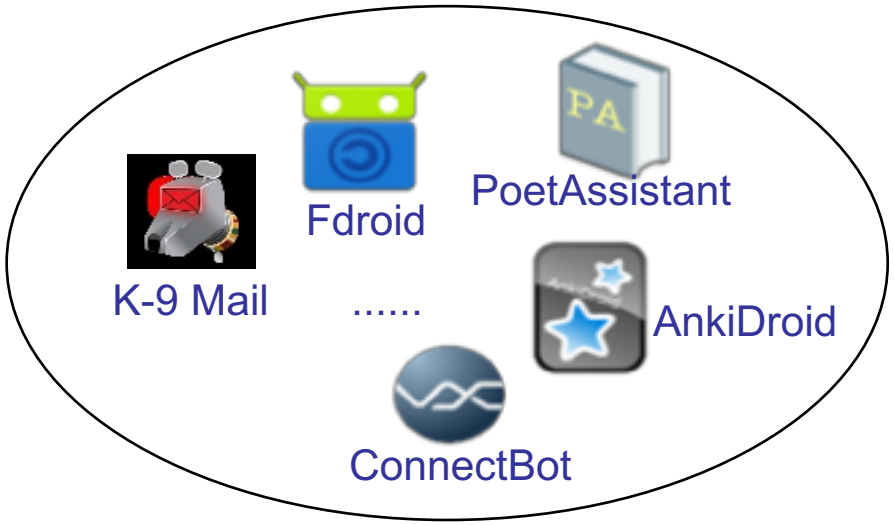
Patch crafted by human

- <17minutes to fix

- 2 days to fix
- Resolved 9 months later

Empirical Evaluation of Droix

Benchmark: *Droixbench*



24 crashes in **15** mobile apps

Evaluation results

- Repaired 12 crashes
- Comparing with developer's repair
  - 1 better ✔
  - 7 syntactic equivalent ✔
  - 1 semantic equivalent ✔
  - 3 UI-behavior equivalent ✔

Summary

- Android repair system that could repair crashes based on a single UI sequence
- Instead of relying on test cases, we use code-level and test-level properties
- Propose a set of lifecycle-aware repair operators
- *Droixbench*: benchmark with reproducible crashes in Android apps, extracted from GitHub