Fixr: Mining and Understanding Bug Fixes for App-Framework Protocol Defects

University of Colorado Boulder

MUSE Site Visit
May 3, 2016
I am not alone
The Android framework is constantly changing

My app seems broken on the new Android 6.0.1. How do I update my app?

How do I leverage the “crowd”?
Explore the bug-fix artifacts created by the crowd (e.g., commits in Github). Find trends in bug-fixing.

Find bug-fix artifacts. Fixes relevant to my code. Fixes that generalize to a “rule” or repair specification.

Apply fixes to my code. Interactively suggest repairs to new code from mined specifications.
Fixr Workflow in Phase 1

- **Inspect the trends** to find common patterns of API usage

- **Inspect the commits** to understand the bug and the fix

**API Usage Trend Analysis**

**Write a query:** snippet of Java code that uses the API in a relevant

**Relevant Commit Search**

**Semantic Search-and-Repair**

**Write a repair specification:** specifies what is the bug and how it can be repaired

**Patched Github**
Fixr Approach:
Mine repair specifications from bug-fixes

Prior Hypothesis of a repair specification
Observe a bug-fix commit

Bayesian Update

Posterior Hypothesis

The Fixr Loop:
Create as many observations as possible
Overview: The Fixr loop

- **Deltar**: Inferring Semantic Deltas and Repair Specifications
- **Prepair**: Deriving Probabilistic Repair Specifications
- **Harvestr**: Social Validation and Mining of Fixes
- **Patchr**: Detecting Potential Bugs and Synthesizing Patches

**FixrDB**

- **symbolic program analysis**
- **numerical-probabilistic program analysis**
- **software engineering for big data**
- **user-centered big data analytics**
- **program synthesis**

**Github**
Fixr Approach: **Commits** are where it starts

- **Need**: Extract features from code commits

- **Github**

- **Features**

- **API Usage Trend Analysis**

- **Relevant Commit Search**

- **Semantic Search-and-Repair**

**Need: Extract features from code commits**
Key Challenge: Finding the “right” feature extraction

“Semantic enough” to minimize noise but “syntactic enough” to be feasible
Fixr Phase 1 Contributions

- Extract commit features at scale
- Find API usage patterns over time
- Index commit feature documents
- Search-and-repair platform for Android apps

Demonstrated the potential of these tools with “simple” features on commits.
Fixr: Phase 1

**Features**: “simple, syntactic” bag of API call differences extracted from commits
Phase 2 Plan: Investigate “richer” features

Main topic for today
Fixr: Phase 1 to Phase 2

**Phase 1**
- Semantic bug condition features
- Mining bug conditions from commits
- Mining event-enabling specifications
- Deriving app-specific event control-flow

**Phase 2**
- Deltar: Inferring Semantic Deltas and Repair Specifications
  - Subgraph isomorphism features
- Prepair: Deriving Probabilistic Repair Specifications
  - Batch feature extraction
- FixrDB
  - API association rule mining
  - Interactive commit search
  - Incremental feature extraction
  - Rule-based patching
  - Harvestr: Social Validation and Mining of Fixes
  - Patchr: Detecting Potential Bugs and Synthesizing Patches

**Features**: “richer, syntactic-semantic” to realize the Fixr loop
Progress towards “Richer” Features

Extract approximate graph isomorphism features

Abstracting event-driven systems with Lifestate Rules
Extract approximate graph isomorphism features
Progress towards “Richer” Features

Extract approximate graph isomorphism features

Abstracting event-driven systems with Lifestate Rules
Abstracting event-driven systems with *Lifestate Rules*
Android is an Event-Driven System

Inter-event **bug fixes** invisible to intra-event ACDFGs

```java
try {
    db.beginTransaction(); ... 
    db.setTransactionSuccessful();
}
finally { db.endTransaction(); }
```

**Callback doInBackground:**
Database operations in the background asynchronously.

**Callback onPostExecute:**
Bug fix: allocate a new MyTxn task for the next transaction.
Android is an Event-Driven System

The event-driven framework uses **callbacks** to notify the application of events and the application uses **callins** to affect how the framework invokes future callbacks.
Android is an Event-Driven System

**Need:** Modeling and reasoning about how **callins** affect **callbacks** (and vice versa)

Exception: Cannot call **execute** on same **AsyncTask** instance.
Contributions

\( \lambda \text{life}: A \) (concrete) model of event-driven systems capturing how callins and callbacks affect each other

Lifestate Rules: A specification language to model the effects of Android callins and callbacks

DroidLife: Mining lifestate specifications and verifying the absence of lifestate “races”
**Fixr: Phase 2**

- **Deltar**: Inferring Semantic Deltas and Repair Specifications
  - Subgraph isomorphism features
- **Prepair**: Deriving Probabilistic Repair Specifications
  - Mining bug conditions from commits
  - Deriving app-specific event control flow
- **FixrDB**: Incremental feature extraction
  - Mining event-enabling specifications
  - Deriving app-specific event control flow
- **Harvestr**: Social Validation and Mining of Fixes
  - Recommending fixes to users
- **Patchr**: Detecting Potential Bugs and Synthesizing Patches
  - Mining fix transformations from isomorphisms

**Features**:
- "richer, syntactic-semantic" to realize the **Fixr** loop
Engagement with the MUSE community

Leidos

Submitted two challenge problems (Web Service Clients and User Interface Regressions)

Corresponding on Phase 2 Infrastructure

Build-ability

Discussion with Leidos, SRI/UCLA, UCI

Collaboration with Rice

Sergio Mover visited in February. On-going discussions.
Fixr Phase 1 Review

Sergio Mover
Can we find the proper bug-fixes in the wild?
Do these hypotheses hold?

API-specific bug-fixes can be transferred to other Apps

We have to reason about commits
Outcome of Phase 1 - Proof of concept

Feature Extraction ➔ API Usage Trend Analysis ➔ Relevant Commit Search ➔ Semantic Search-and-Repair

Inspect the trends to find common patterns of API usage
Inspect the commits to understand the bug and the fix

Demonstrates the feasibility of the project
Overview of the results and challenges from Phase 1

Results

Feature Extraction

API Usage Trend Analysis

Challenges

Relevant Commit Search

Semantic Search-and-Repair

Github

Patched
Fixr: Feature Extraction

- Feature Extraction
- Github
- API Usage Trend Analysis
- Relevant Commit Search
- Semantic Search-and-Repair
- Patched
The features describe a change narrowed to the Android APIs.
Does the feature extraction scale?

Data Extraction

Github

16K repos with 510K code commits

2.83M changed files in 3 hours

125M code features in 28 hours

Commit Feature
android.Button
android.Dialog
...

1.39M commit-relationships in 4 hours
Phase 2 challenges

What if some **new** data is available?

What if we need to extract a **new feature**?

Need a flexible pipeline: incremental and robust to add new types of features
Fixr: API Usage Trend Analysis

Feature Extraction → API Usage Trend Analysis → Relevant Commit Search → Semantic Search-and-Repair
What is the correct usage of the API?

Code examples of Android SQLite transactions that I've seen appear to automatically call `db.setTransactionSuccessful()` right before `db.endTransaction()`.

I am wondering if that is actually best practice or whether there should be some conditional check before calling `db.setTransactionSuccessful()`.

The canonical pattern for transactions:

```java
beginTransaction();
+ y {
    //db operations ...
    setTransactionSuccessful();
} finally {
    endTransaction();
}
```

“Recipe” exchanged by discussion on forums

Find patterns of usage for the Android APIs
When the antecedents change in a commit, the consequent also changes.

Frequent association rules describe common patterns of API usage.
When does a commit violate a rule?

- The consequent does not change in the commit.

A violation may indicate the presence of a bug.

Real bug found mining our corpus.

Code snippet from the GitHub project fanvoudroid/

```java
void syncUsers(List<User> users) {
    SQLiteDatabase mDb = mOpenHelper.getWritableDatabase();
    beginTransaction();
    for (User u : users) {
        if (existsUser(u.getId())) {
            updateUser(u);
        } else {
            createUserInfo(u);
        }
    }
    endTransaction();
    mDb.setTransactionSuccessful();
}
```
Does a rule capture a bug-fix?

The same bug-fix is applied in a short period of time.

Idea: Analyze the “time-signature” of about 5000 commits.

The bug-fix to the API happens at the “peaks.”
Can we mine API patterns?

Manual classification:
- API best practices
- API required usages
- Accidental Rules
## Most frequent rules

<table>
<thead>
<tr>
<th>Rule Antecedents =&gt; Consequent</th>
<th>Matching Commits</th>
<th>Violating Commits</th>
<th>Potential Bugs</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>API best practices</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>hasNext, iterator =&gt; next</td>
<td>9024</td>
<td>225</td>
<td>0</td>
</tr>
<tr>
<td>getString, makeText =&gt; show</td>
<td>3727</td>
<td>154</td>
<td>0</td>
</tr>
<tr>
<td>commit, getSupportFragmentManager =&gt; beginTransaction</td>
<td>3526</td>
<td>43</td>
<td>0</td>
</tr>
<tr>
<td>getContentResolver, moveToFirst =&gt; query</td>
<td>2302</td>
<td>76</td>
<td>0</td>
</tr>
<tr>
<td><strong>API required usages</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>beginTransaction, setTransactionSuccessful =&gt; endTransaction</td>
<td>1544</td>
<td>78</td>
<td>4</td>
</tr>
<tr>
<td>obtainStyleAttributes =&gt; recycle</td>
<td>1208</td>
<td>30</td>
<td>7</td>
</tr>
<tr>
<td>setContextText, setSmallIcon =&gt; setContentTitle</td>
<td>1337</td>
<td>13</td>
<td>6</td>
</tr>
<tr>
<td>setContentIntent, setContentTitle =&gt; setSmallIcon</td>
<td>1024</td>
<td>27</td>
<td>6</td>
</tr>
<tr>
<td><strong>Accidental Rules</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>setTitle, setMessage, setNegativeButton =&gt; setPositiveButton</td>
<td>1340</td>
<td>70</td>
<td>0</td>
</tr>
<tr>
<td>getMeasuredHeight, layout =&gt; getMeasuredWidth</td>
<td>1525</td>
<td>42</td>
<td>0</td>
</tr>
<tr>
<td>getMeasuredWidth, layout =&gt; getMeasuredHeight</td>
<td>1525</td>
<td>42</td>
<td>0</td>
</tr>
<tr>
<td>getBottom, getLeft =&gt; getRight</td>
<td>1115</td>
<td>54</td>
<td>0</td>
</tr>
</tbody>
</table>

- Rules divided by classification
- Higher number of matching commits
- Manual inspection of 60 violating commits
Challenges for phase 2

```java
try {
    db.beginTransaction();
    // db operations
    db.setTransactionSuccessful();
} finally {
    db.endTransaction();
}
```

**Correct pattern of usage:**
- **order of method calls**

**GREEN LINES**: lines added in the new version

**SAME METHOD CALLS IN THE COMMIT**: the bag of methods feature for the commit is **empty**!

**Bags of method calls are control and data flow agnostic: need better features**
Fixr: Relevant Commit Search

1. Feature Extraction
2. API Usage Trend Analysis
3. Relevant Commit Search
4. Semantic Search-and-Repair

Diagram showing the flow from Github to relevant commit search to patched output.
How should I change my code?

Oh, I have in my code:

```java
MediaScannerConnection.scanFile(context, p, m, l)
```

Find **commits** that change a "similar" code
Can we find relevant bugs and fixes?

Find commits with **similar features** to the queries

**Query**

```
Java
MediaScannerConn.scanFile(this)
```

**List of commits**

- `MediaScanner...`
- `Context.getApp...`

Searched for 5 pattern queries
### Results of the RCS evaluation

<table>
<thead>
<tr>
<th>Bug Pattern</th>
<th>Commits Retrieved (num)</th>
<th>Bugs in First 10 (num)</th>
<th>Fixes in First 10 (num)</th>
</tr>
</thead>
<tbody>
<tr>
<td>View.setTag</td>
<td>19</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MediaScannerConnection.scanFile</td>
<td>11</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>getSystemService(CAMERA_SERVICE)</td>
<td>139</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>GoogleApiClient.Builder</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>MediaPlayer.setDataSource</td>
<td>21</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Queries for 5 known bug

Out of 1.39M commit-relations, bugs and fixes in the first 10 results can be found in the first 10 bugs and fixes in the first 10 results with feature-based document search.
Challenges for phase 2

Commit fix: inverts the order of the operations

QUERY
$db$.endTransaction();
$db$.beginTransaction();

Relevant Commit Search

COMMIT
-db$.endTransaction();
$db$.beginTransaction();
+db$.endTransaction()

Same set of method calls in the commit: the bag of methods for the commit is empty!

Bags of method calls are control and data flow agnostic: need better features
Fixr: Semantic Search-and-Repair

Feature Extraction → API Usage Trend Analysis → Relevant Commit Search → Semantic Search-and-Repair
Can we transfer the bug-fix?

The Repair Specification describes:
- a semantic bug condition
- a fix transformation of the source code

Given a repair specification, **find and repair** the buggy code in the corpus.
Search-and-repair platform for Android apps

- Github
- Relevant Commits
- Static Analysis
- Syntactic Patch

Relevant Commit Search

+ repair specification

Static Analysis

+ patch locations

Syntactic Patch

+ patch locations
Are API repairs applicable “in the wild”? 

- 16K repos with 510K code commits
- 48 repos with relevant commits
- 3 repair specifications
- 48 repos with relevant commits
- 22 repos buildable at HEAD
- 22 repos buildable at HEAD
- 6 alarms in 5 repos with 5 bugs
- 16K repos with 510K code commits
- 5 repos patchable
- 5 repos patchable
- Relevant Commit Search
- Static Analysis
- Synthesis: compiles and loads without crash
Challenges for Phase 2

Finding and repairing bugs in the wild is feasible...

...but now we encode the repair specification manually

Automatically synthesize bug conditions from commits
Fixr: Summary of Phase 1

- Found 1600 rules and 25 real bugs
- Successfully found and fixed 5 bugs in the corpus
- API-specific bug-fixes can be transferred to other repositories
- Commits are the right artifact for:
  - mining API usage patterns
  - finding bug-fixes

Feature Extraction ➔ API Usage Trend Analysis ➔ Relevant Commit Search ➔ Semantic Search-and-Repair

Github ➔ Patched

Extracted 1.39M commits from 16K of repositories

Found bugs and fixes in the first 10 results of the search
Successfully found and fixed 5 bugs in the corpus

Commits are the right artifact for:
- mining API usage patterns
- finding bug-fixes

API-specific bug-fixes can be transferred to other repositories
Fixr: Challenges for Phase 2

- Extract and use features that are aware of the control and data flow.
- Graph features and graph isomorphism.

Feature Extraction → API Usage Trend Analysis → Relevant Commit Search → Semantic Search-and-Repair

Infer semantic bug conditions from the corpus.
Fixr Phase 2 Preview
Fixr: Phase 1 to Phase 2

Phase 2

- **Deltar**: Inferring Semantic Deltas and Repair Specifications
  - Subgraph isomorphism features
  - API association rule mining
  - Interactive commit search
  - Incremental feature extraction

- **Prepair**: Deriving Probabilistic Repair Specifications
  - Batch feature extraction
  - Rule-based patching

- **FixrDB**: Software mining for big data
  - Batch feature extraction
  - Rule-based patching

- **Harvestr**: Social Validation and Mining of Fixes
  - Recommending fixes to users

- **Patchr**: Detecting Potential Bugs and Synthesizing Patches

Features: “richer, syntactic-semantic” to realize the Fixr loop
Fixr: Phase 2

**Deltar**: Inferring Semantic Deltas and Repair Specifications
- Semantic bug condition features
- Mining bug conditions from commits

**Prepair**: Deriving Probabilistic Repair Specifications
- Deriving app-specific event control flow
- Probabilistic repair specification

**Harvestr**: Social Validation and Mining of Fixes
- Subgraph isomorphism features
- Mining bug conditions from commits
- Semantic facts

**Patchr**: Detecting Potential Bugs and Synthesizing Patches
- Incremental feature extraction
- Mining bug conditions from commits
- Mining fix transformations from isomorphisms

**FixrDB**
- Incremental feature extraction

**Features**: “richer, syntactic-semantic” to realize the Fixr loop
Subgraph isomorphisms better characterize essential syntactic changes.
Fixr: Phase 2

Deltar: Inferring Semantic Deltas and Repair Specifications

Prepair: Deriving Probabilistic Repair Specifications

FixrDB

Harvestr: Social Validation and Mining of Fixes

Patchr: Detecting Potential Bugs and Synthesizing Patches

Features: “richer, syntactic-semantic” to realize the Fixr loop

Semantic bug condition features

Mining bug conditions from commits

Mining event-enabling specifications

Deriving app-specific control-flow

numerical-probabilistic program analysis

Subgraph isomorphism features

Ware mining for big data

repair specification

semantic facts

probabilistic repair specification

Incremental feature extraction

Mining bug conditions from commits

Mining fix transformations from isomorphisms

FixrDB

GitHub

interaction

commit

Recommend fixing fixes to users

Features: ‘richer, syntactic-semantic’ to realize the Fixr loop

semantic statistical-social

syntactic

social

Program analysis

Program synthesis

Feature: ‘richer, syntactic-semantic’ to realize the Fixr loop

Big data

Software engineering for big data

Analytics

GitHub

Program analysis

Numerical-probabilistic program analysis

Feature: ‘richer, syntactic-semantic’ to realize the Fixr loop

Syntactic

Statistical-social

Semantic
Need: Infer a **bug condition** characterizing reachable states derived from the change.
Diff in terms of reachable states

Challenge: Approximate removed states and the relationship to the “underlying” bug condition
## Fixr: Phase 2

### 2016

<table>
<thead>
<tr>
<th>Month</th>
<th>Activities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mar</td>
<td>Richer Feature Extraction</td>
</tr>
<tr>
<td>Jun</td>
<td>Supporting Richer Feature Extraction</td>
</tr>
<tr>
<td>Sep</td>
<td>Interacting with Fixr Users</td>
</tr>
<tr>
<td>Dec</td>
<td>Infrastructure Support for Big Code</td>
</tr>
</tbody>
</table>

### 2017

<table>
<thead>
<tr>
<th>Month</th>
<th>Activities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mar</td>
<td>Richer Feature Extraction</td>
</tr>
<tr>
<td>Jun</td>
<td>Supporting Richer Feature Extraction</td>
</tr>
<tr>
<td>Sep</td>
<td>Interacting with Fixr Users</td>
</tr>
<tr>
<td></td>
<td>Infrastructure Support for Big Code</td>
</tr>
</tbody>
</table>

### Activities

- **Richer Feature Extraction**
  - Graph Isomorphism Algorithms
  - API Usage Understanding with Graphs Isomorphisms
  - Graph Isomorphisms for Fix Transformations
  - Defining and Inferring Semantic Bug Conditions
  - Generalizing Bug Conditions on Multiple Commits
- **Supporting Richer Feature Extraction**
  - Enabling Rule Mining
  - Deriving App-Specific Control-Flow
  - Android App-Specific Semantic Feature Extraction
  - Active Learning of Lifestate Automata
- **Interacting with Fixr Users**
  - Decision Support for Bug-Fix Commits
  - Decision Support for Bug-Fix Specifications
- **Infrastructure Support for Big Code**
  - Incremental Extraction of Features
  - Android App-Specific Semantic Feature Extraction in the Large
Today’s Agenda

<table>
<thead>
<tr>
<th>Time</th>
<th>Session</th>
<th>Presenter(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>12:30-12:45</td>
<td><strong>Introduction</strong></td>
<td>Bor-Yuh Evan Chang</td>
</tr>
<tr>
<td>12:45-1:15</td>
<td><strong>Phase 1 Review</strong></td>
<td>Sergio Mover</td>
</tr>
<tr>
<td>1:15-1:30</td>
<td><strong>Phase 2 Preview</strong></td>
<td>Bor-Yuh Evan Chang</td>
</tr>
<tr>
<td>1:30-2:10</td>
<td><strong>Richer Feature Extraction</strong></td>
<td>Sriram Sankaranarayanan</td>
</tr>
<tr>
<td>2:10-2:30</td>
<td>Break</td>
<td></td>
</tr>
<tr>
<td>2:30-3:10</td>
<td><strong>Supporting Richer Feature Extraction</strong></td>
<td>Shawn Meier, PhD Student</td>
</tr>
<tr>
<td>3:10-3:30</td>
<td><strong>Active Learning for Lifestate Automata</strong></td>
<td>Krishna Chaitanya Sripada, MS Student</td>
</tr>
<tr>
<td>3:30-3:50</td>
<td><strong>Interacting with Fixr Users</strong></td>
<td>Ryo Suzuki, PhD Student</td>
</tr>
<tr>
<td>3:50-4:10</td>
<td><strong>Infrastructure Support for Big Code</strong></td>
<td>Kenneth M. Anderson</td>
</tr>
<tr>
<td>4:10-4:30</td>
<td><strong>Discussion</strong></td>
<td></td>
</tr>
</tbody>
</table>
Commits + Time Series = Trends

Semantic Repair Specification

Android API Evolution