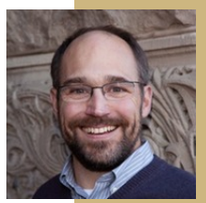
Fixr: Mining and Understanding Bug Fixes for App-Framework Protocol Defects

Bor-Yuh Evan Chang



Kenneth M. Anderson

Pavol Černý

Sriram Sankaranarayanan

Tom Yeh

Sergio Mover

Shawn Meier

Krishna Chaitanya Sripada

Ryo Suzuki

Rhys Braginton Pettee Olsen

Maxwell Russek

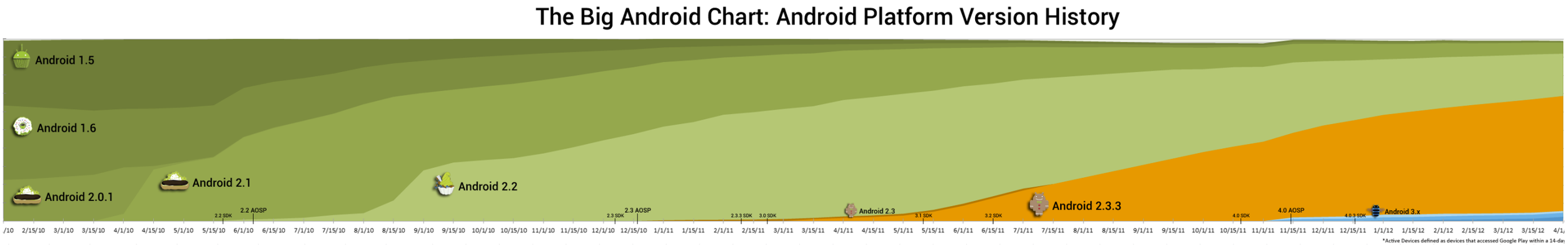
University of Colorado Boulder

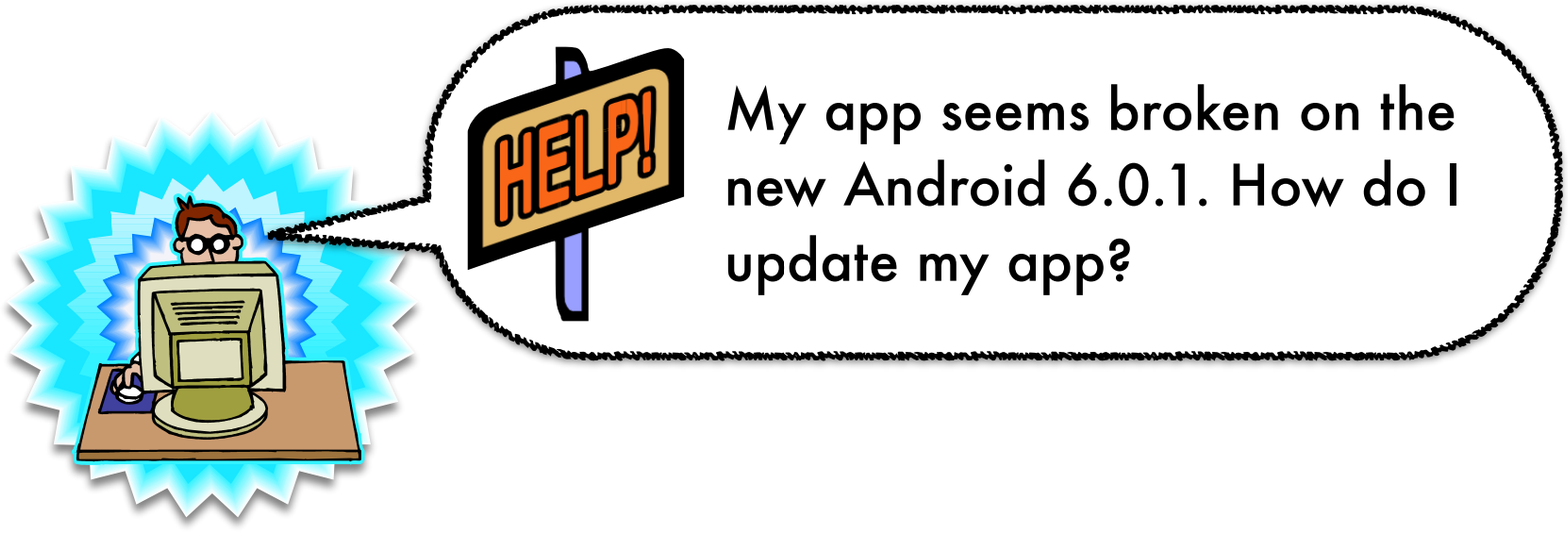
PLV

MUSE Site Visit
May 3, 2016

I am not alone

The Android framework is constantly changing

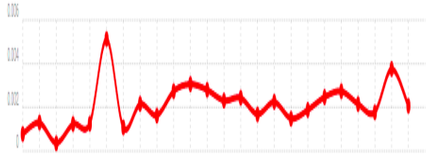




How do I leverage the "crowd"?

Fixr Goals and Tools

API Usage Trend Analysis



Explore the bug-fix artifacts created by the crowd (e.g., commits in Github). Find trends in bug-fixing.

Relevant Commit Search

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

Semantic Search-and-Repair

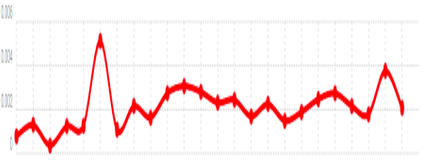
Apply fixes to my code. Interactively suggest repairs to new code from mined specifications

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



Relevant Commit Search

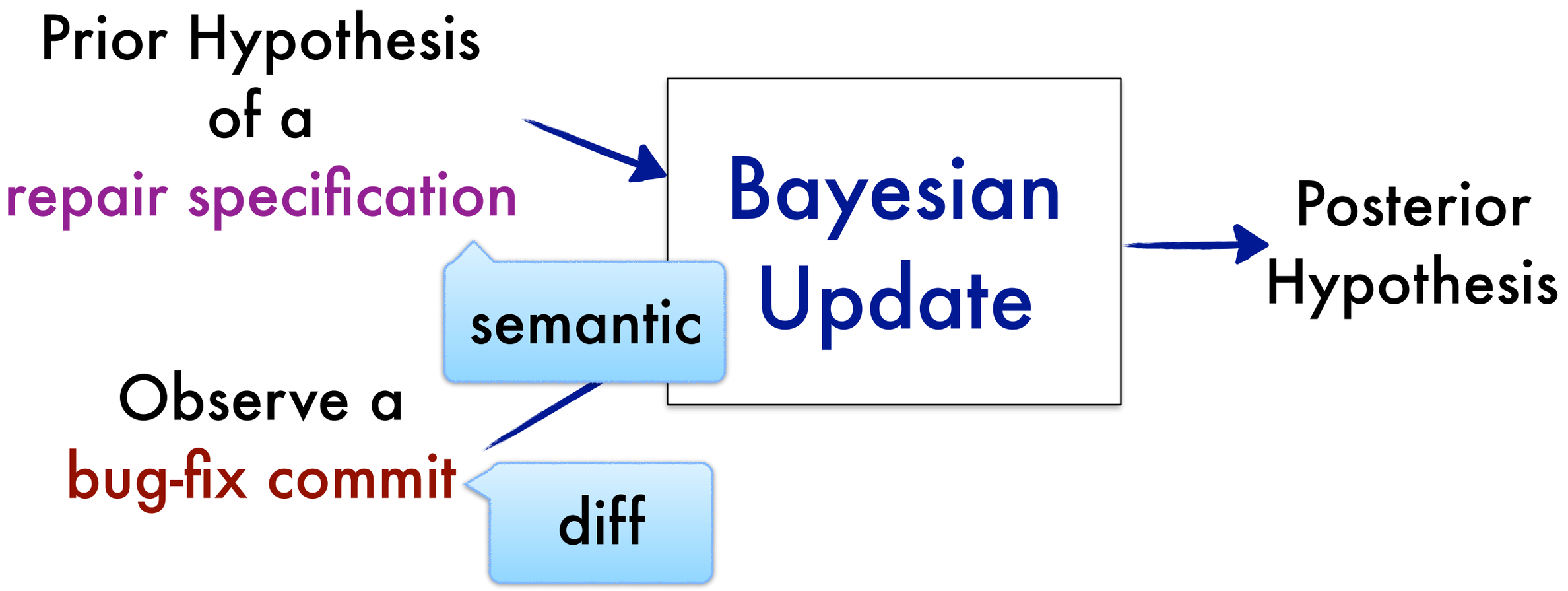
Semantic Search-and-Repair

Patched Github

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

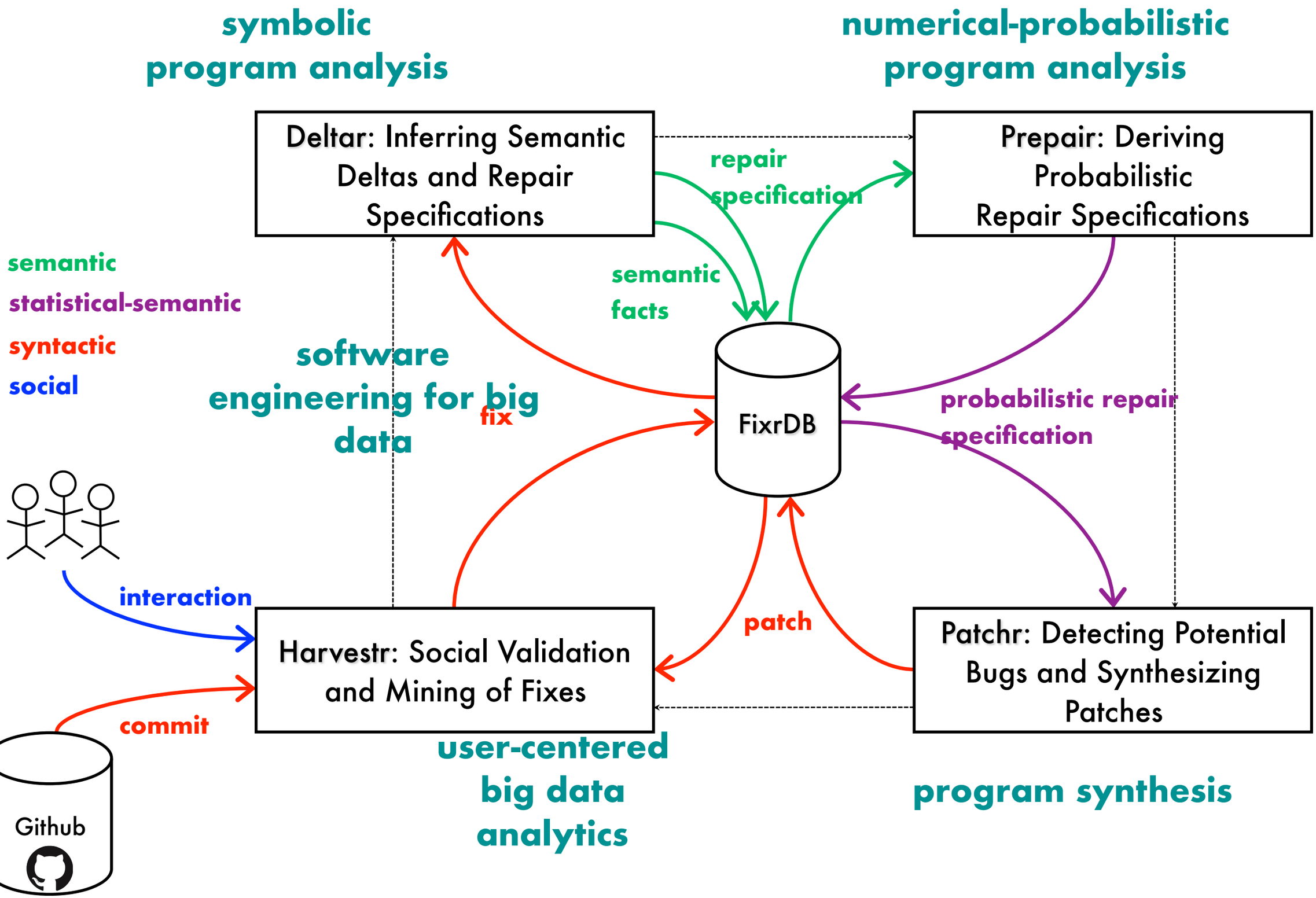
Write a repair specification: specifies what is the bug and how it can be repaired

Fixr Approach:
Mine repair specifications from bug-fixes

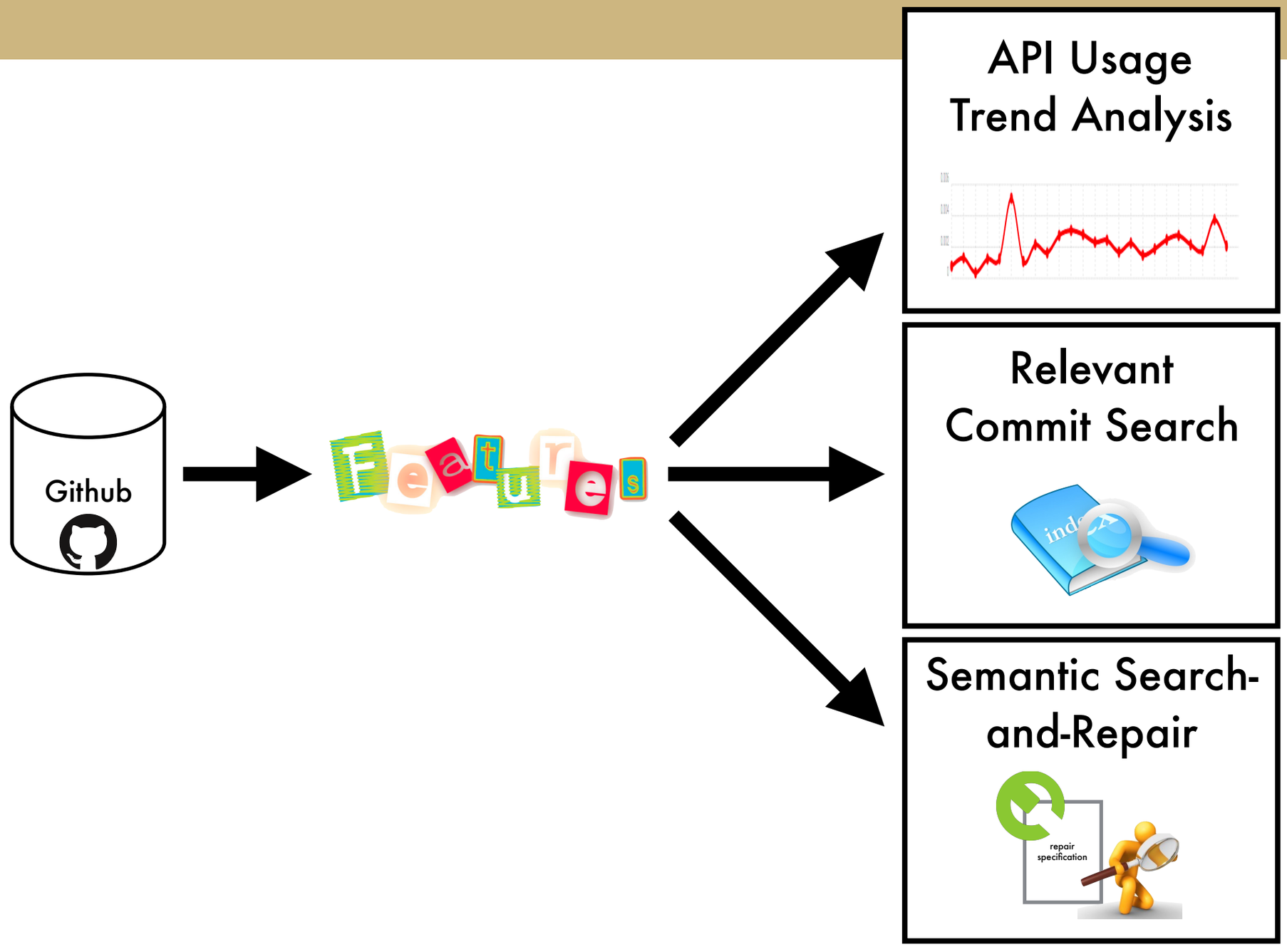


The **Fixr** Loop:
Create as many observations as possible

Overview: The **Fixr** loop



Fixr Approach: Commits are where it starts

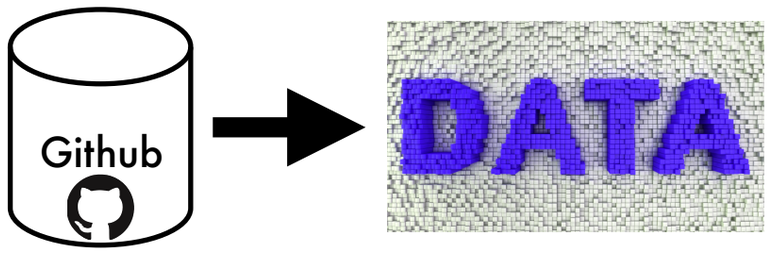


Need: Extract features from code commits

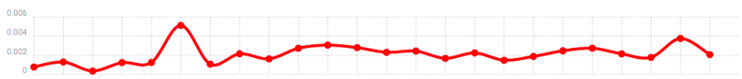
Key Challenge: Finding the "right" feature extraction

"Semantic enough" to minimize noise but "syntactic enough" to be feasible

Fixr Phase 1 Contributions



Extract commit features at scale



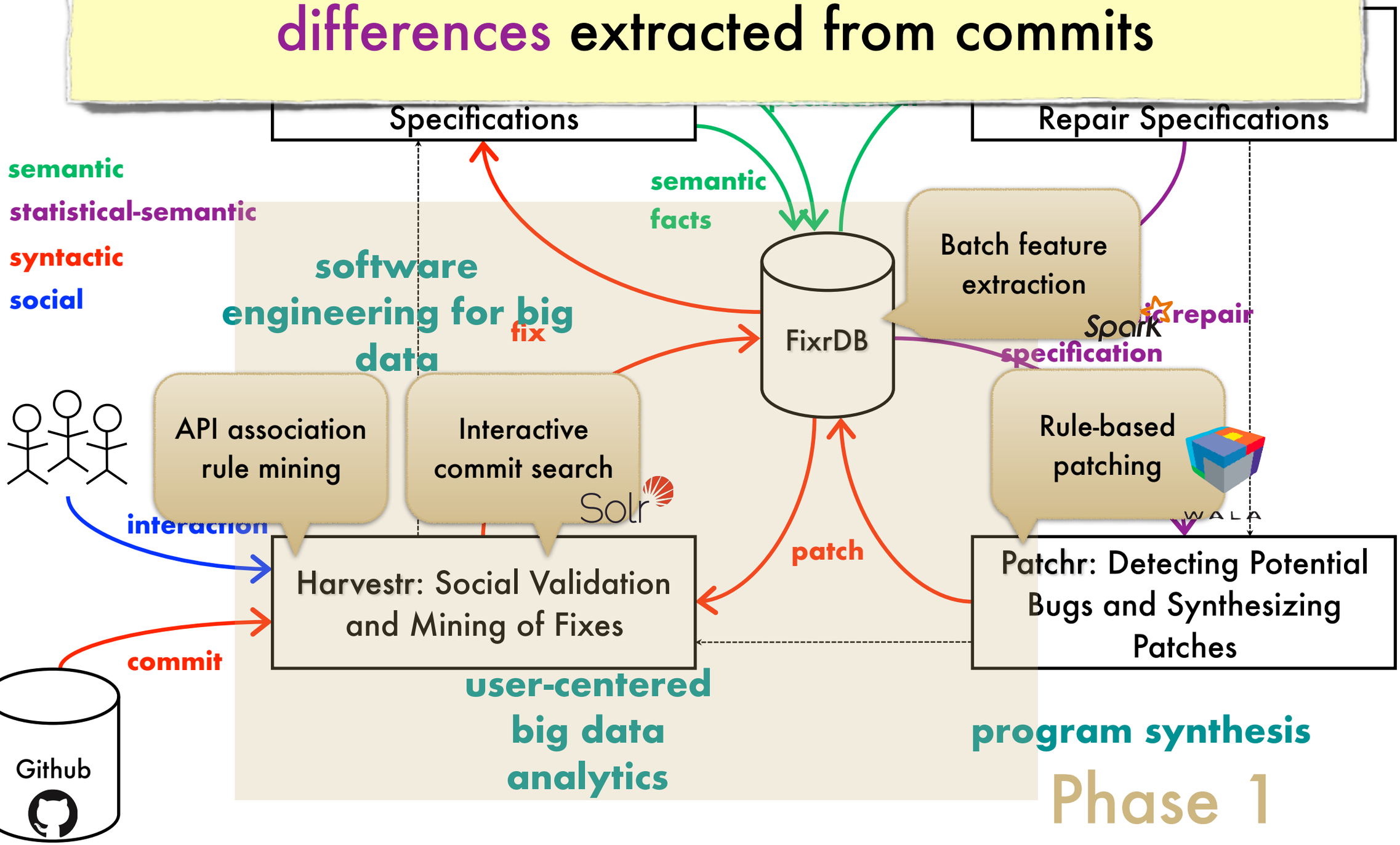
Find API us time

Demonstrated the potential of these tools with "simple" features on commits

Se platform for A apps

Fixr: Phase 1

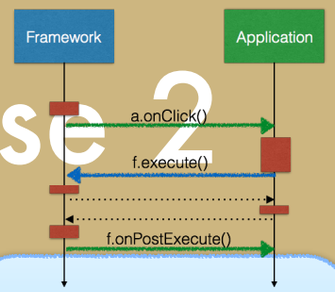
Features: "simple, syntactic" bag of API call differences extracted from commits



Phase 2 Plan: Investigate "richer" features

Main topic for today

Fixr: Phase 1 to Phase 2



Semantic bug condition features

Mining bug conditions from commits

Mining event-enabling specifications

Deriving app-specific event control-flow

symbolic program analysis

numerical-probabilistic program analysis

Phase 2

semantic

statistical-s

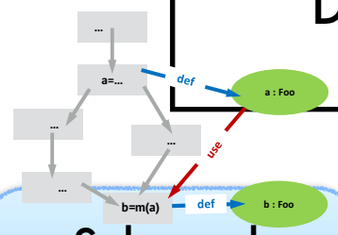
syntactic

social

Deltar: Inferring Semantic Deltas and Repair Specifications

repair specification

Prepair: Deriving Probabilistic Repair Specifications



Subgraph isomorphism features

semantic facts

Batch feature extraction

Mining fix transformations from isomorphisms

ware

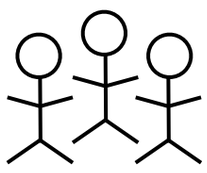
ing for big data

fix

FixrDB

Spark

repair specification



API association rule mining

Interactive commit search

Solr

Incremental feature extraction

Rule-based patching

WALA

interaction

patch

Harvestr: Social Validation and M

Patchr: Detecting Potential

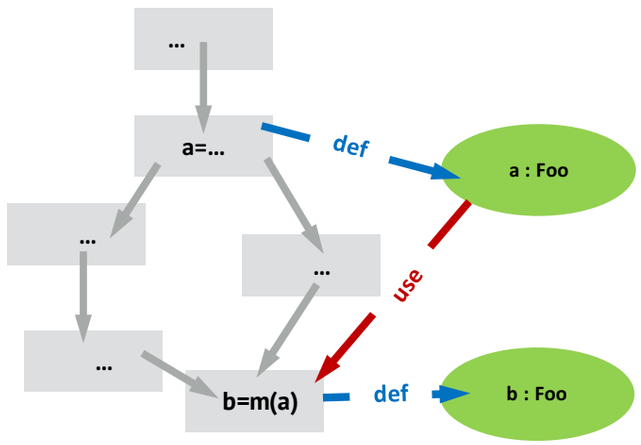
commit

Recommending fixes to users

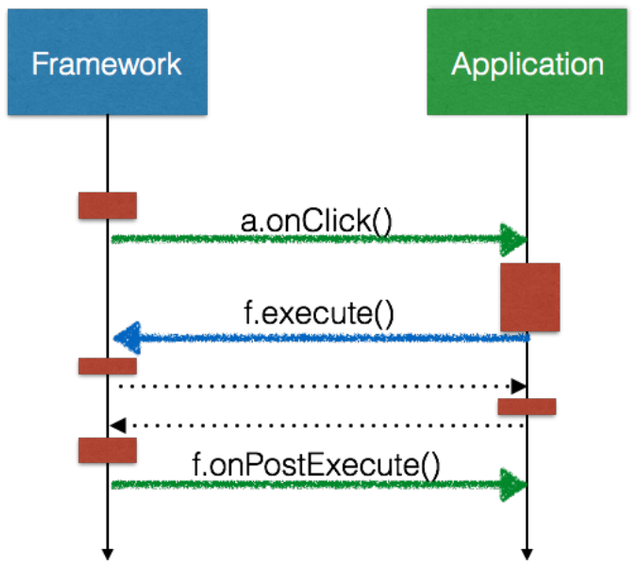
Github

Features : "richer, syntactic-semantic" to realize the **Fixr** loop

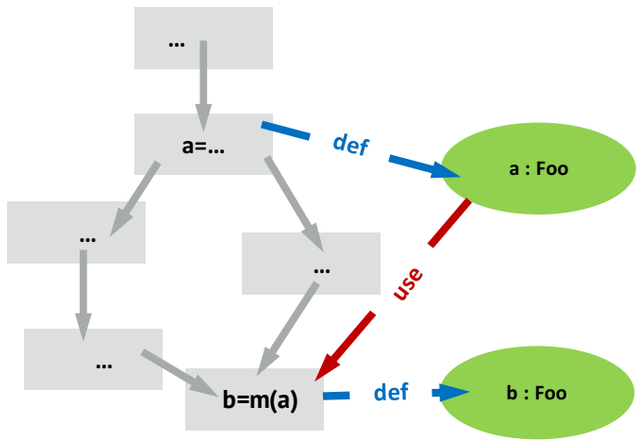
Progress towards "Richer" Features



Extract approximate graph isomorphism features

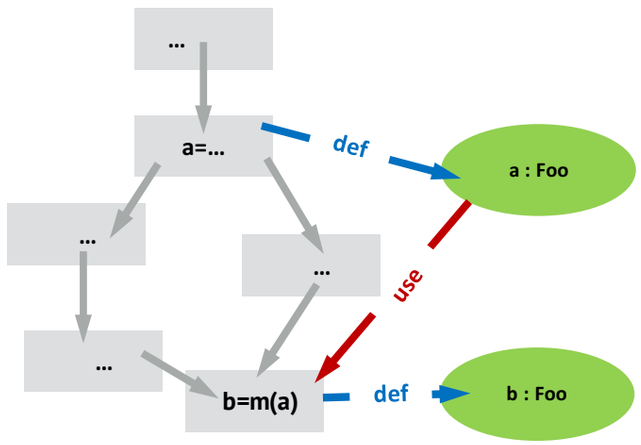


Abstracting event-driven systems with *Lifestate Rules*

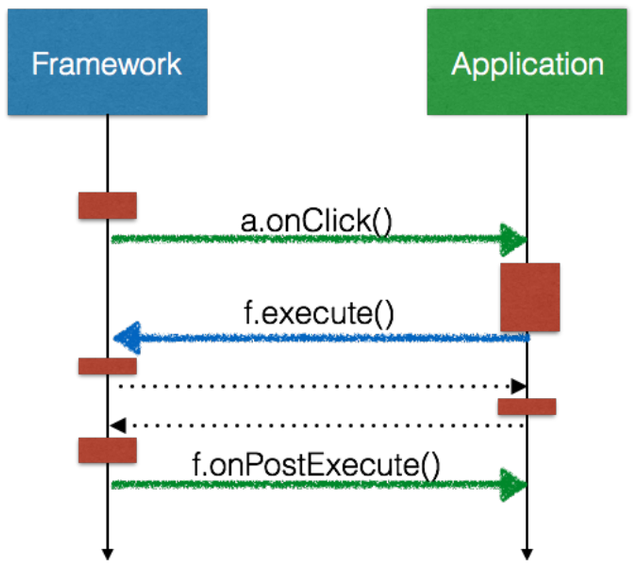


Extract approximate graph isomorphism features

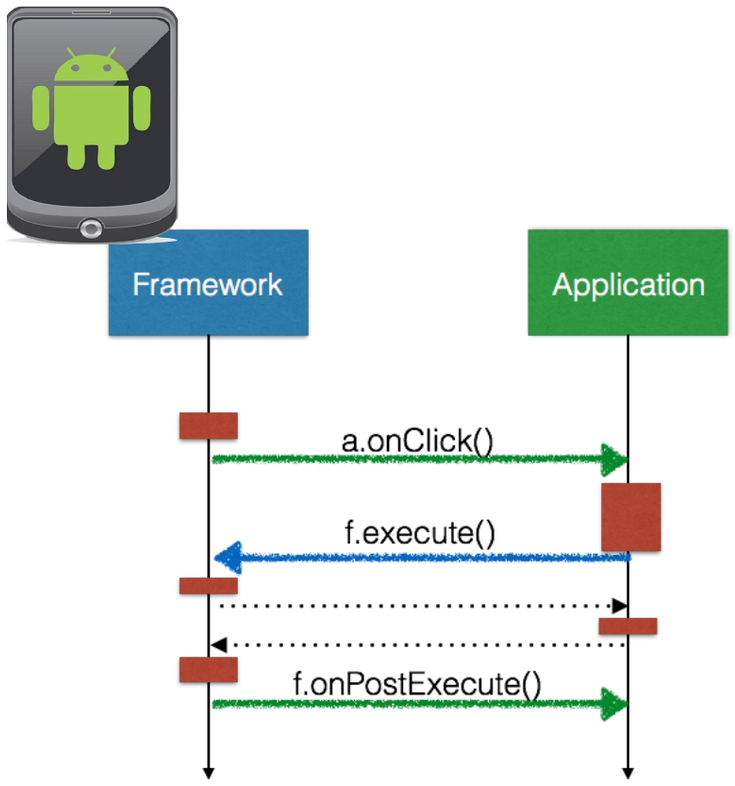
Progress towards "Richer" Features



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

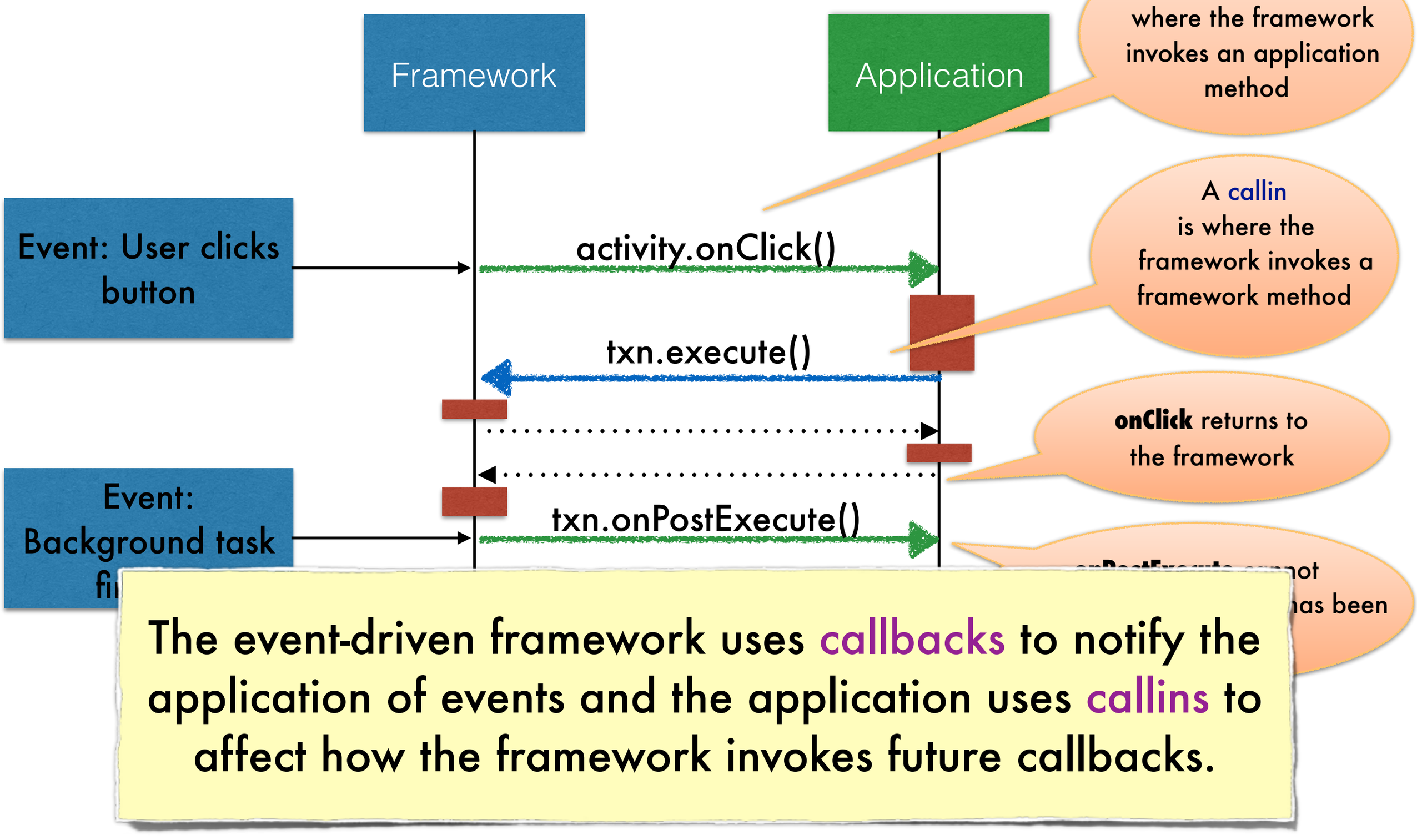
triggered on user click.

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

Callback doInBackground:
Database operations in the background asynchronously.

Add callback onPostExecute:
Bug fix: allocate a new MyTxn task for the next transaction.

Android is an Event-Driven System

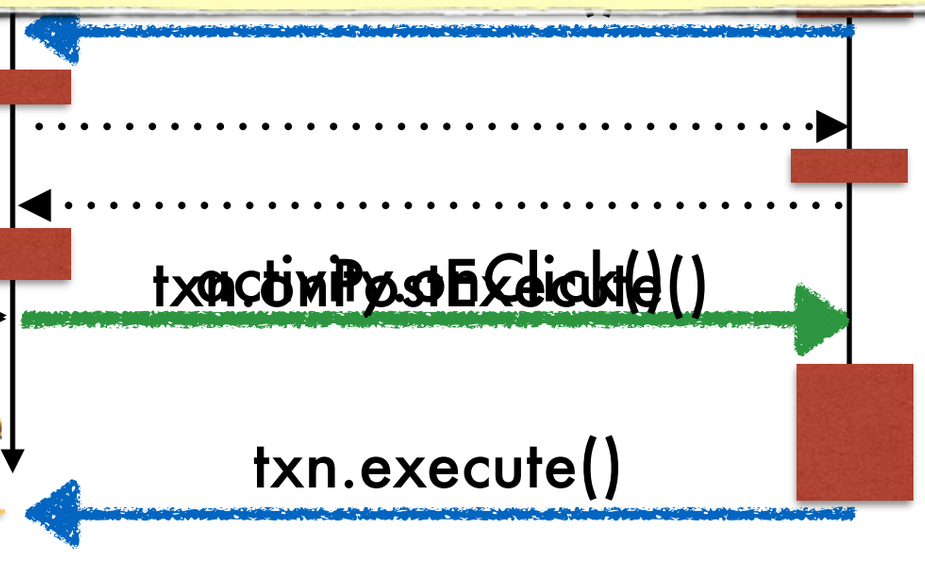


The event-driven framework uses callbacks to notify the application of events and the application uses callins to affect how the framework invokes future callbacks.

Android is an Event-Driven System

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

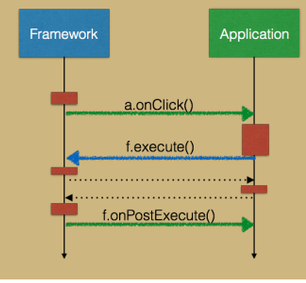
Event: Us
butt



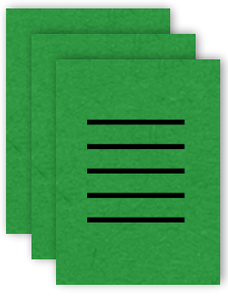
Event: Background task finishes

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

Contributions



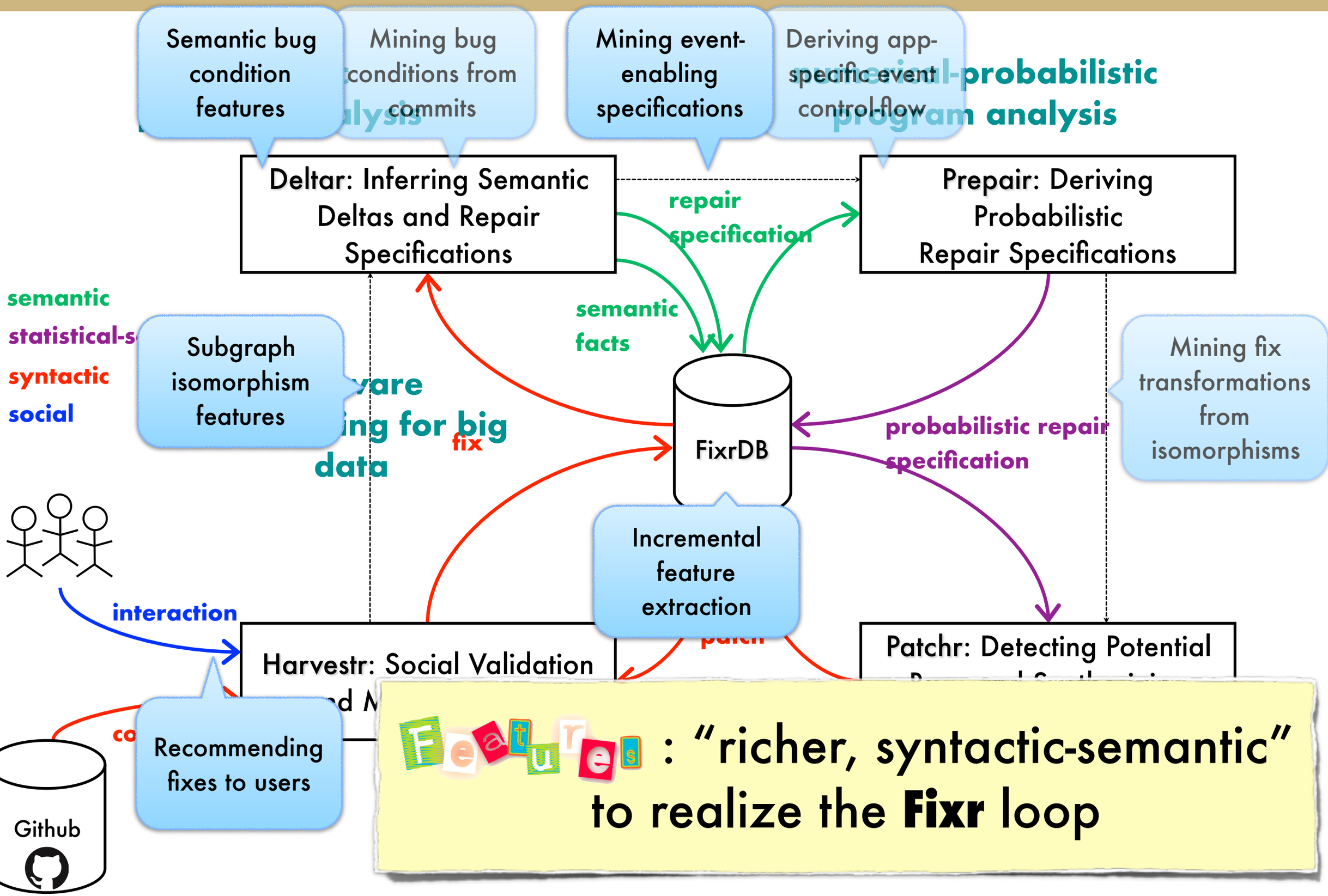
λife: A (concrete) model of event-driven systems capturing how callins and callbacks affect each other



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


Semantic bug condition features
Mining bug conditions from commits
Mining event-enabling specifications
Deriving app-specific event control-flow
numerical-probabilistic program analysis
Deltar: Inferring Semantic Deltas and Repair Specifications
repair specification
Prepair: Deriving Probabilistic Repair Specifications
semantic
statistical-s
syntactic
social
Subgraph isomorphism features
semantic facts
FixrDB
probabilistic repair specification
Mining fix transformations from isomorphisms
fix
data
Incremental feature extraction
interaction
Harvestr: Social Validation
Patchr: Detecting Potential
Recommending fixes to users
Github
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

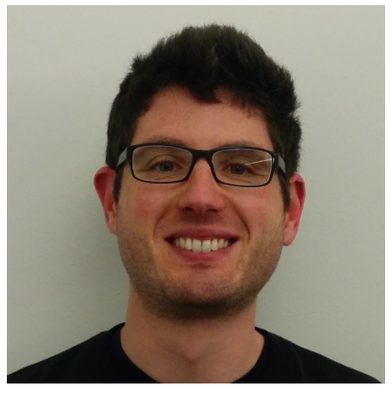
Discussion with Leidos, SRI/UCLA, UCI

Collaboration with Rice

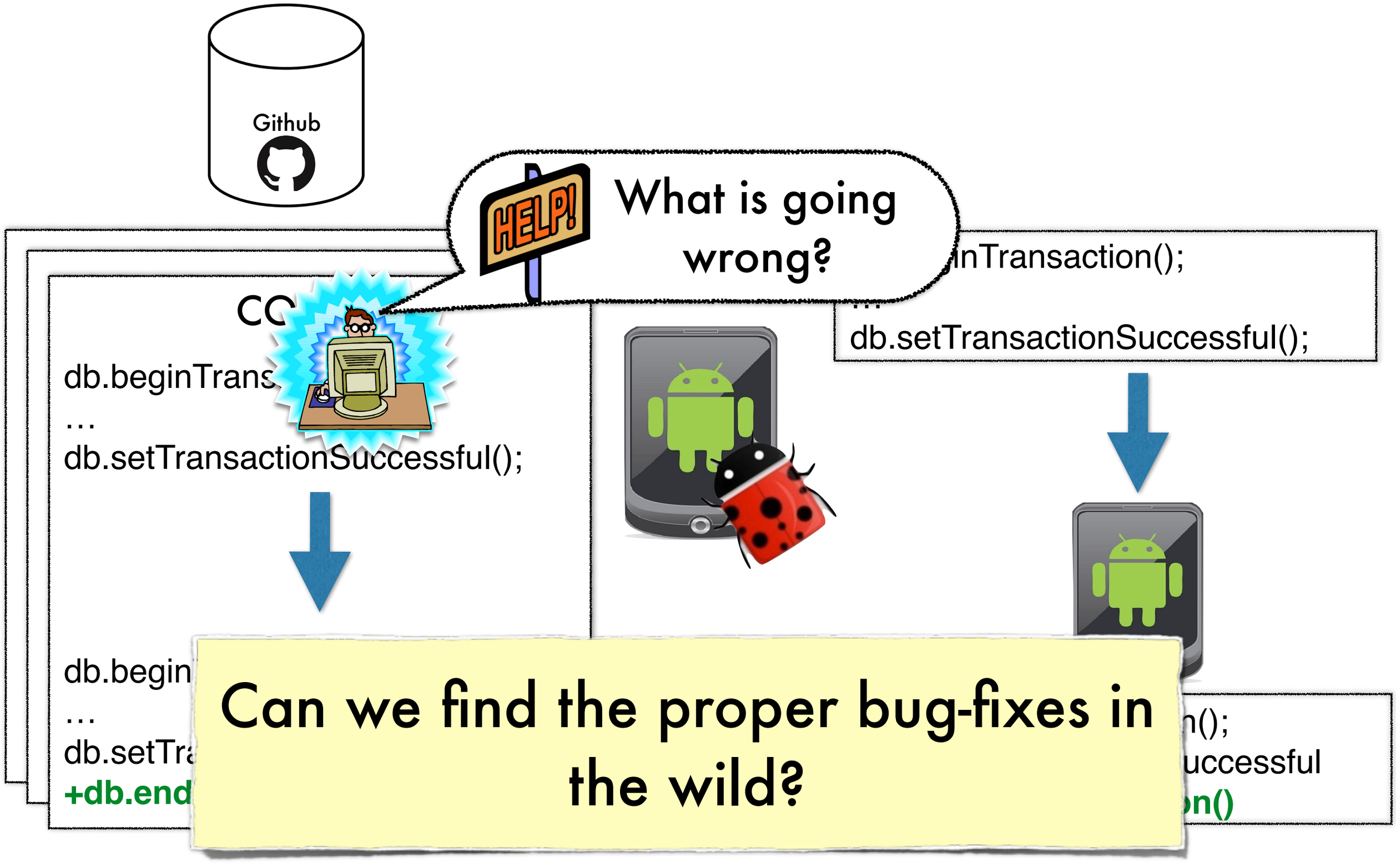
Sergio Mover visited in February. On-going discussions.

Fixr Phase 1 Review

Sergio Mover

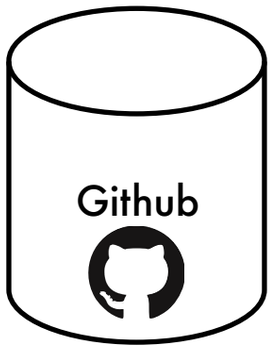


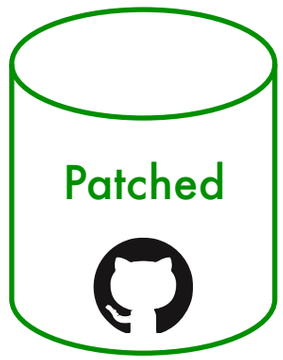
Fixr problem



Fixr - Main hypotheses

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





COMMIT

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

We have to reason about commits

Do these hypotheses hold?

Outcome of Phase 1 - Proof of concept

Github

Inspect the trends to find common patterns of API usage

Inspect the commits to understand the bug and the fix

Patched

Feature Extraction

API Usage Trend Analysis

Relevant Commit Search

Semantic Search-and-Repair

Demonstrates the feasibility of the project

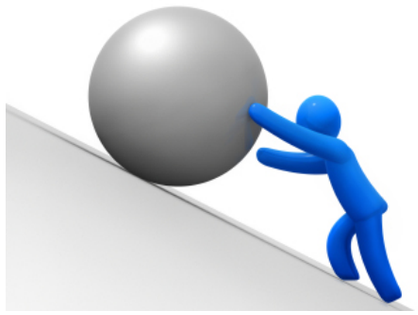
air

es what is

can be

Overview of the results and challenges from Phase 1

Results



Challenges

Feature Extraction

API Usage Trend Analysis

Relevant Commit Search

Semantic Search-and-Repair

Fixr: Feature Extraction

Feature Extraction

API Usage Trend Analysis

Relevant Commit Search

Semantic Search-and-Repair

"API-specific" features from commits

COMMIT

-import android.Bu
import android.Datab
import android.Viev
+import android. ialog;
…
db.beginTransaction();
+myapp.myMethod();
+db.endTransaction();
+view.setTag(this,id);
-view.setTag(this);

Only Android specific features

Imports that changed

Github

Feature Extraction

API-specific Features

Bag of imports

android.Button
android.Dialog

Bag of methods

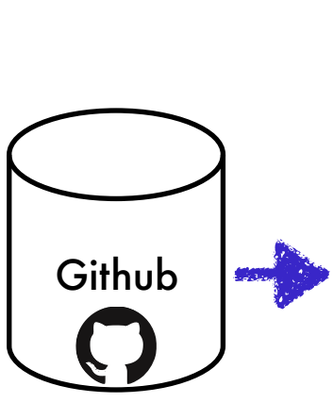
Database.endTransaction(void)
View.setTag(Object)
View.setTag(Object, int)

Method calls that changed

-RED LINES: lines removed
+GREEN LI

The features describe a change narrowed to the Android APIs

Does the feature extraction scale?



Data Extraction

Commit Feature

android.Button

android.Dialog

...

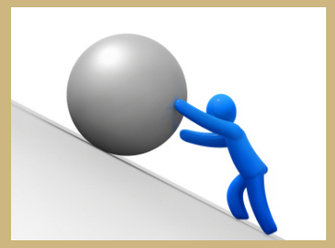
16K repos with 510K code commits

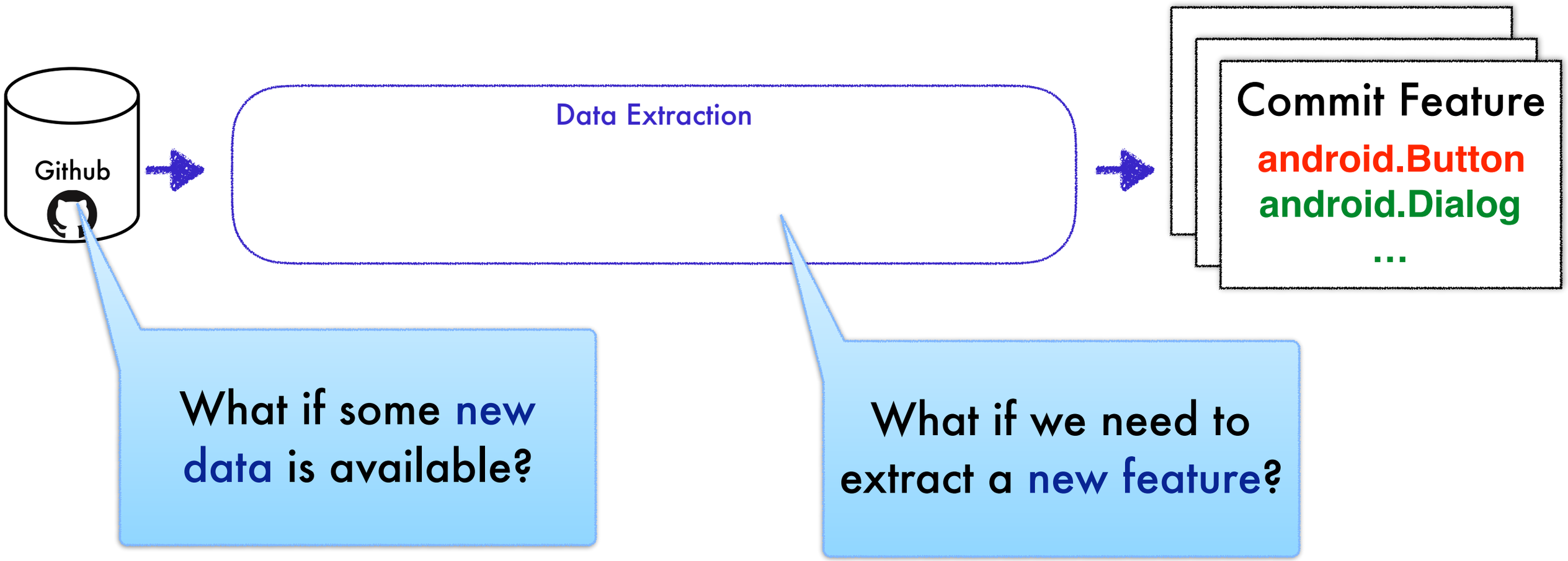
2.83M changed files in 3 hours

125M code features in 28 hours

1.39M commit-relations in 4 hours

Phase 2 challenges





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?

3

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()`.

How should I use thee API?

5

The canonical pattern for transactions:

```
beginTransaction();
try {
    //db operations ...

    setTransactionSuccessful();
} finally {
    endTransaction();
}
```

"Recipe" exchanged by discussion on forums

Find patterns of usage for the Android APIs

Frequent Association Rule Mining

Frequent rule

Strength: **0.950519**

Matches/Violating: **1556** / **81**

Antecedents

setTransactionSuccessful

Consequent

endTransaction

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?

Matches/Violating: **1556** / **81**

The consequent does not change in the commit

Antecedents

setTransactionSuccessful

Consequent

endTransaction

A violation may indicate the presence of a bug

Real bug found mining our corpus

from the GitHub project fanvoudroid/

```
1107        oid syncUsers(List<com.ch_linghu.fanfoudroid.data.User> users){
1108        teDatabase mDb = mOpenHelper.getWritableDatabase();
1109        beginTransaction();
1110        for(com.ch_linghu.fanfoudroid.data.User u:users){
1111            if(existsUser(u.id)){
1112                updateUser(u);
1113            }else{
1114                createUserInfo(
1115            }
1116        }
1117        mDb.setTransactionSuccessful();
1118    }
```

Missing call to endTransaction

Does a rule capture a bug-fix?

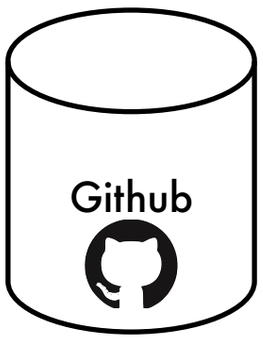
The same bug-fix is applied in a short period of time

Idea: Analyze the "time-signature" of

about 5000 commits

The bug-fix to the API happens at the "peaks"

Can we mine API patterns?



COMMIT

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

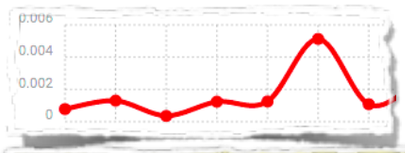
API Usage Trend Analysis

COMMIT

-db.endTran();
db.beginTran;
+db.endTrans n()

Rules

antecedents => consequent



1600 frequent rules

Manual classification:

- API best practices
- API required usages
- Accidental Rules

Most frequent rules

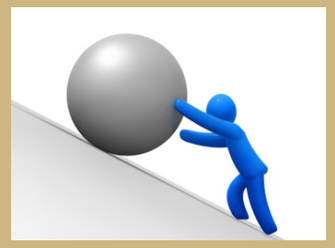
Rule Antecedents => Consequent	Matching Commits	Violating Commits	Potential Bugs
API best practices			
hasNext, iterator => next	9024	225	0
getString, makeText => show	3727	154	0
commit, getSupportFragmentManager => beginTransaction	3526	43	0
getContentResolver, moveToFirst => query	2302	76	0
API required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

Rules divided by classific

Highe

Manual inspection of 60 violating commits

Challenges for phase 2



COMMIT

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

+GREEN LINES: lines added in the new version

Correct pattern of usage:

- **order of method calls**

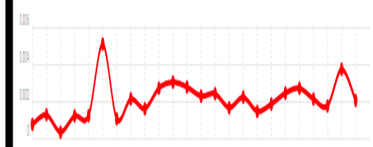
Same method calls in the commit: the bag of methods feature for the commit is empty!

Bags of method calls are control and data flow agnostic: need better features

Fixr: Relevant Commit Search

Feature Extraction

API Usage Trend Analysis



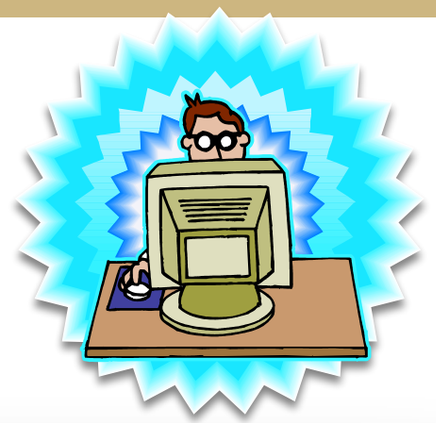
Relevant Commit Search

Semantic Search-and-Repair

How should I change my code?

Oh, I have in my code:

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



Query.java — /Users/Admin/Documents/muse/muse_repos/FixrRelevantCodeSearch/evaluation/queries/MediaScannerConnection

Query.java

```
1  import android.media.MediaScannerConnection;
2
3  public class Query extends AppCompatActivity implements MediaS
4
5      private MediaScannerConnection mConnection;
6
7      protected void onCreate() {
8        MediaScannerConnection.scanFile(this,
9                                        new String[] { "" },
10                                       null,
11                                       null);
12     }
13 }
14
```

[6] fasteque/rgb-tool

android-rgb-tool/src/main/java/com/fastebro/androidrgbtool/ui/MainActivity.java

Commit: @3a93e80 - fixed AOSP binder leak

Child: @3a93e80, Parent: @7ed5fc8

Diff or Source Code

```
             * Tell the media scanner about the new file so that it
             * immediately available to the user.
             */
-            MediaScannerConnection.scanFile(this,
+            MediaScannerConnection.scanFile(getApplicationContext(),
                     new String[]{event.photoPath}, null,
```

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)

Searched for 5 pattern queries

Relevant Commit Search

List of commits

-

MediaScanner…

+

Context.getApp…

Results of the RCS evaluation

Bug Pattern	Commits Retrieved (num)	Bugs in First 10 (num)	Fixes in First 10 (num)
View.setTag	19	1	1
MediaScannerConnection.scanFile	11	5	2
getSystemService(CAMERA_SERVICE)	139	1	0
GoogleApiClient.Builder	2	1	0
MediaPlayer.setDataSource	21	0	0

Queries for 5 known bug

out of 1.39/ commit-relati

bugs and fixes in

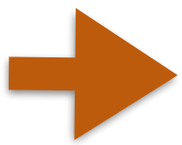
Bugs and fixes can be found in the first 10 results with feature-based document search

Challenges for phase 2

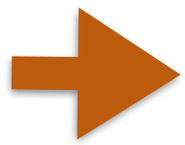
Commit fix: inverts the order of the operations

QUERY

db.endTransaction();
db.beginTransaction();



Relevant Commit Search



COMMIT

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

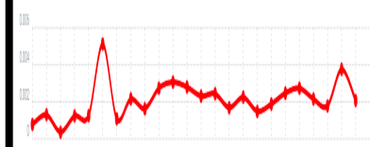
Same set of method calls in the commit: the bag of methods for the commit is empty!

Bags of method calls are control and data flow agnostic: need better features

Fixr: Semantic Seach-and-Repair

Feature Extraction

API Usage Trend Analysis



Relevant Commit Search

Semantic Search-and-Repair

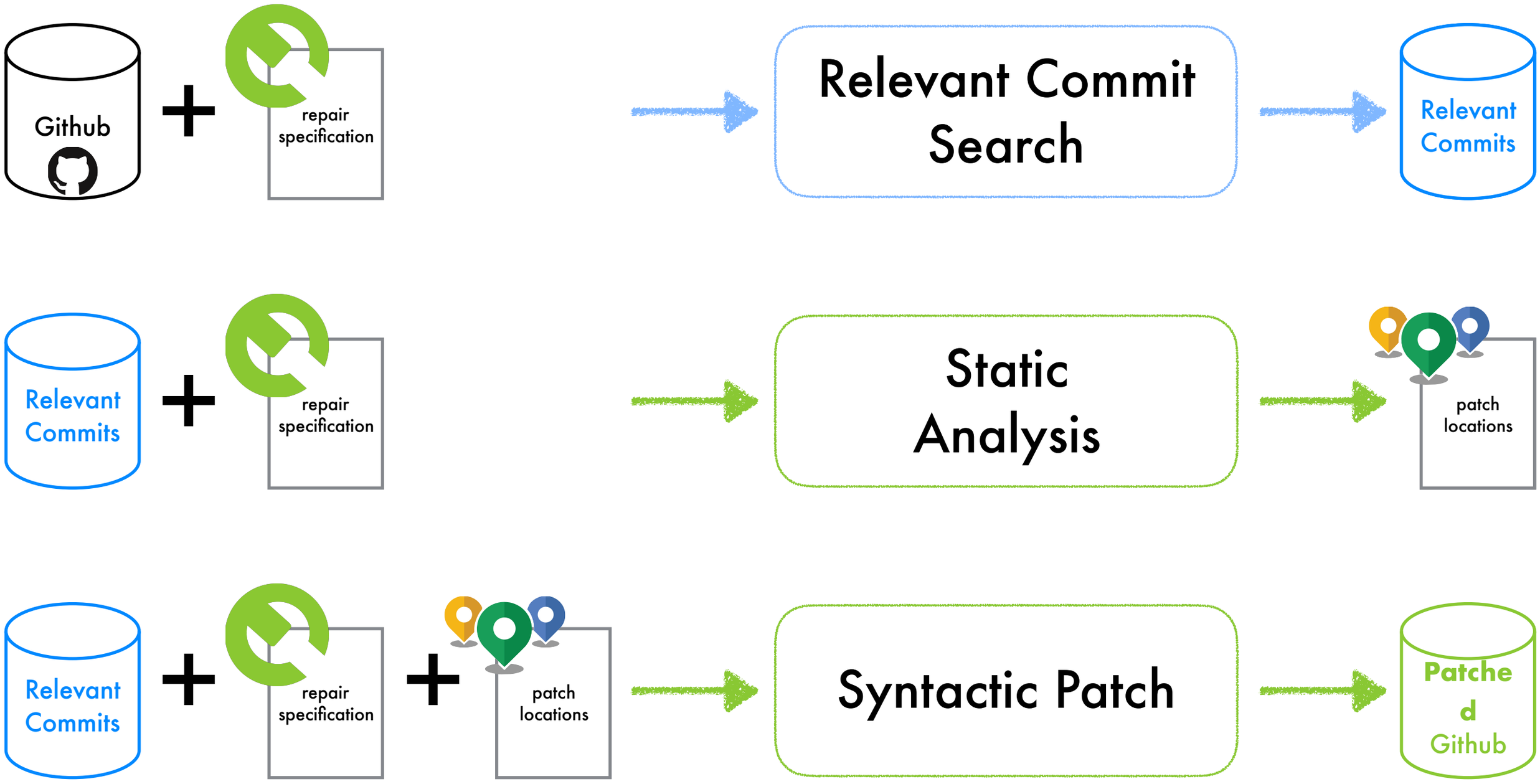
Can we transfer the bug-fix?

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



repairs applicable "in the wild"?

16K repos with 510K code commits

Github

+

3 repair specifications

Relevant Commit Search

48 repos with relevant commits

Relevant Commits

+

repair specification

22 repos buildable at HEAD

Static Analysis

6 alarms in 5 repos with 5 bugs

Relevant Commits

+

repair specification

+

patch locations

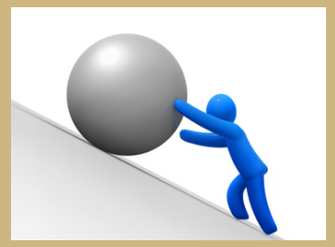
Syn

compiles and loads without crash

Patche d

5 repos patchable

Challenges for Phase 2

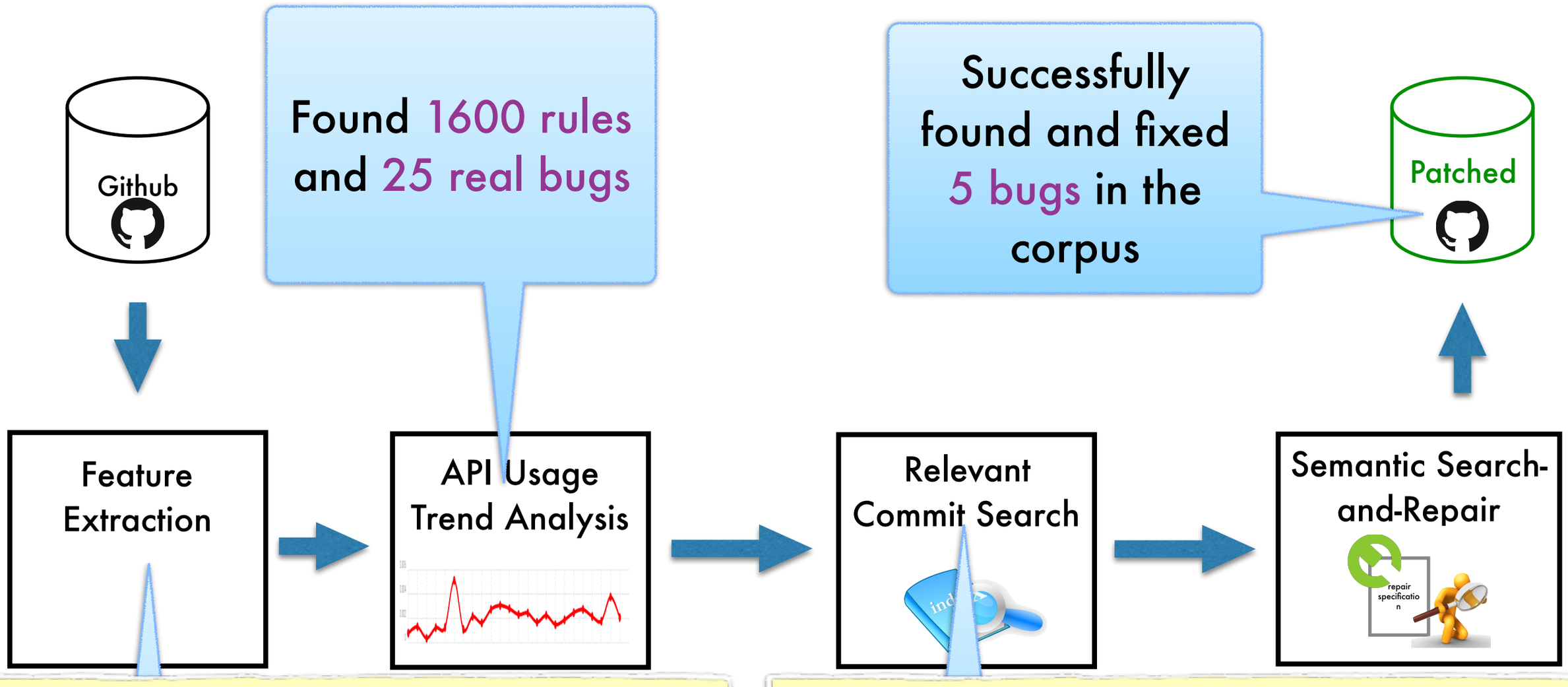


Finding and repairing bugs in the wild is feasible…

…but now we encode the repair specification manually

Automatically synthesize bug conditions from commits

Fixr: Summary of Phase 1



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

Commits are the right artifact for:

- mining API usage patterns
- finding bug-fixes

Fixr: Challenges for Phase 2

Extract and use features that are aware of the control and data flow

Graph features and graph isomorphism

Github

Patched

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

semantic

statistical-s

syntactic

social

Semantic bug condition features

Mining bug conditions from commits

Mining event-enabling specifications

Deriving app-specific event control-flow

numerical-probabilistic program analysis

Deltar: Inferring Semantic Deltas and Repair Specifications

repair specification

Prepair: Deriving Probabilistic Repair Specifications

semantic facts

Subgraph isomorphism features

ware

ing for big data

fix

FixrDB

Batch feature extraction

Spark

repair specification

Mining fix transformations from isomorphisms

API association rule mining

Interactive commit search

Solr

Incremental feature extraction

Rule-based patching

WALA

interaction

Harvestr: Social Validation

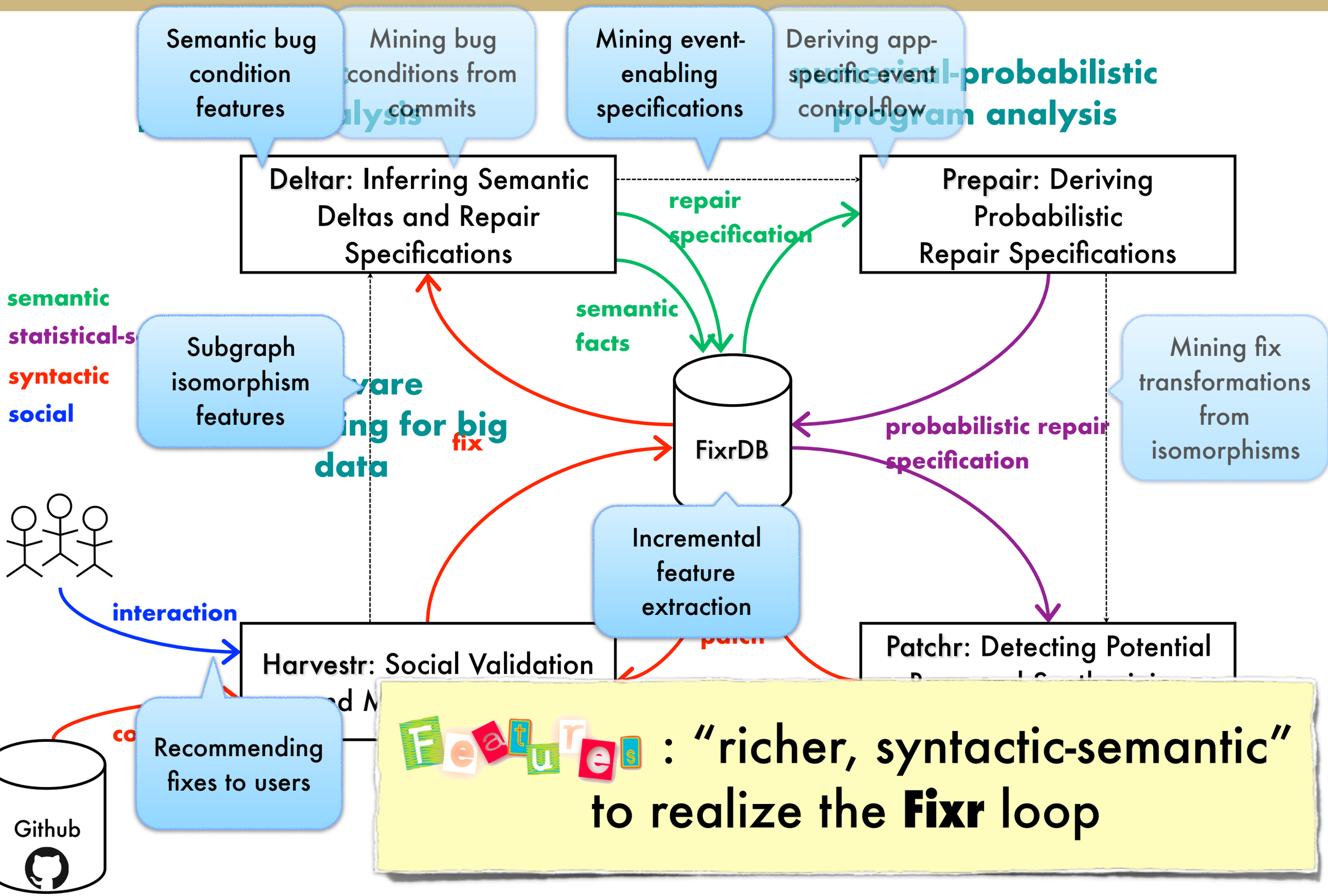
Patchr: Detecting Potential

Recommending fixes to users

Github

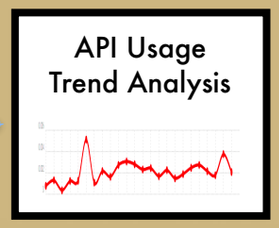
Features : "richer, syntactic-semantic" to realize the **Fixr** loop

Fixr: Phase 2


Semantic bug condition features
Mining bug conditions from commits
Mining event-enabling specifications
Deriving app-specific event control-flow
numerical-probabilistic program analysis
Deltar: Inferring Semantic Deltas and Repair Specifications
repair specification
Prepair: Deriving Probabilistic Repair Specifications
semantic
statistical-s
syntactic
social
Subgraph isomorphism features
semantic facts
FixrDB
probabilistic repair specification
Mining fix transformations from isomorphisms
fix
data
Incremental feature extraction
interaction
Harvestr: Social Validation
Patchr: Detecting Potential
Recommending fixes to users
Github
Features : "richer, syntactic-semantic" to realize the **Fixr** loop

Extract features that expose relevant syntactic changes

Subgraph isomorphism features

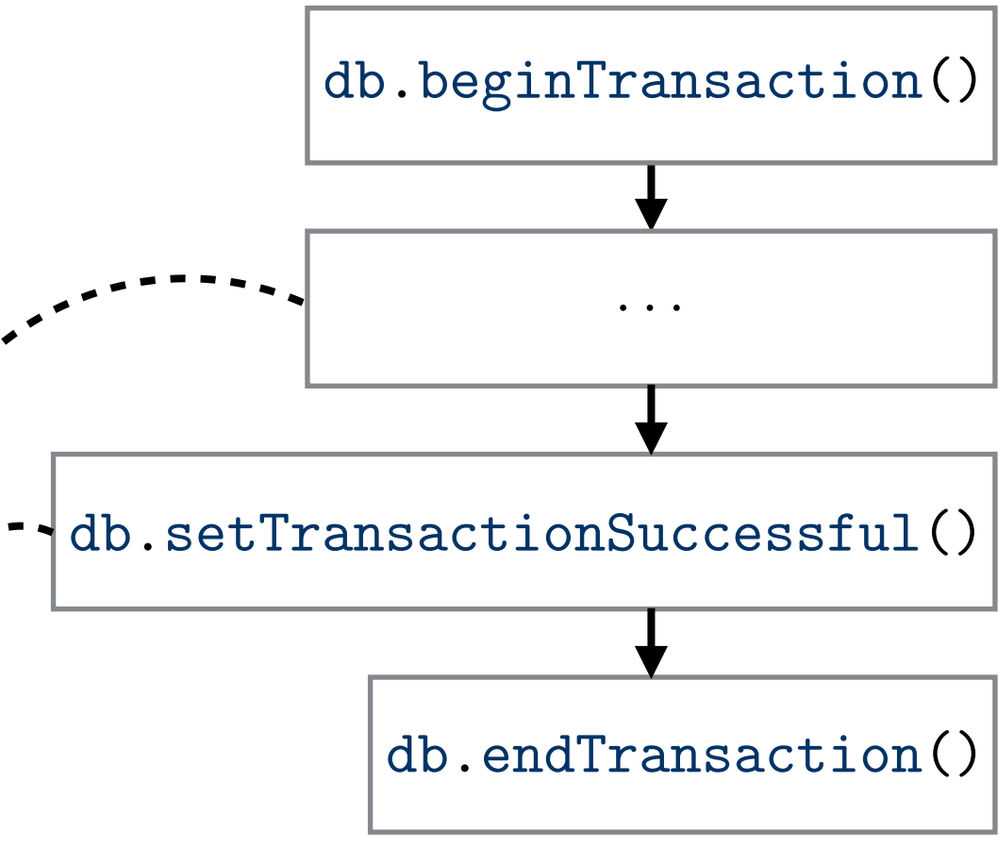


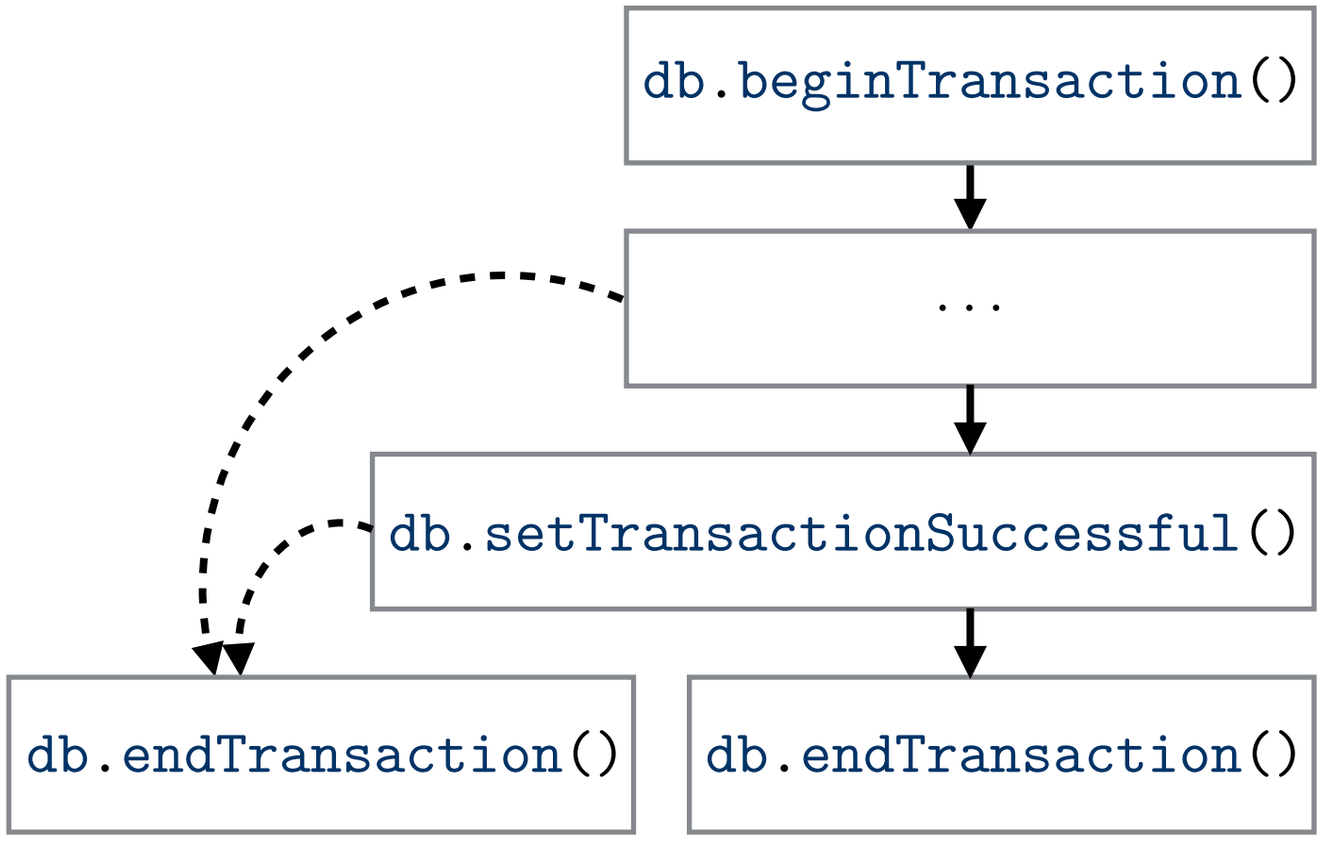
The same set of API calls

```
db.beginTransaction()
...
db.
db.
```

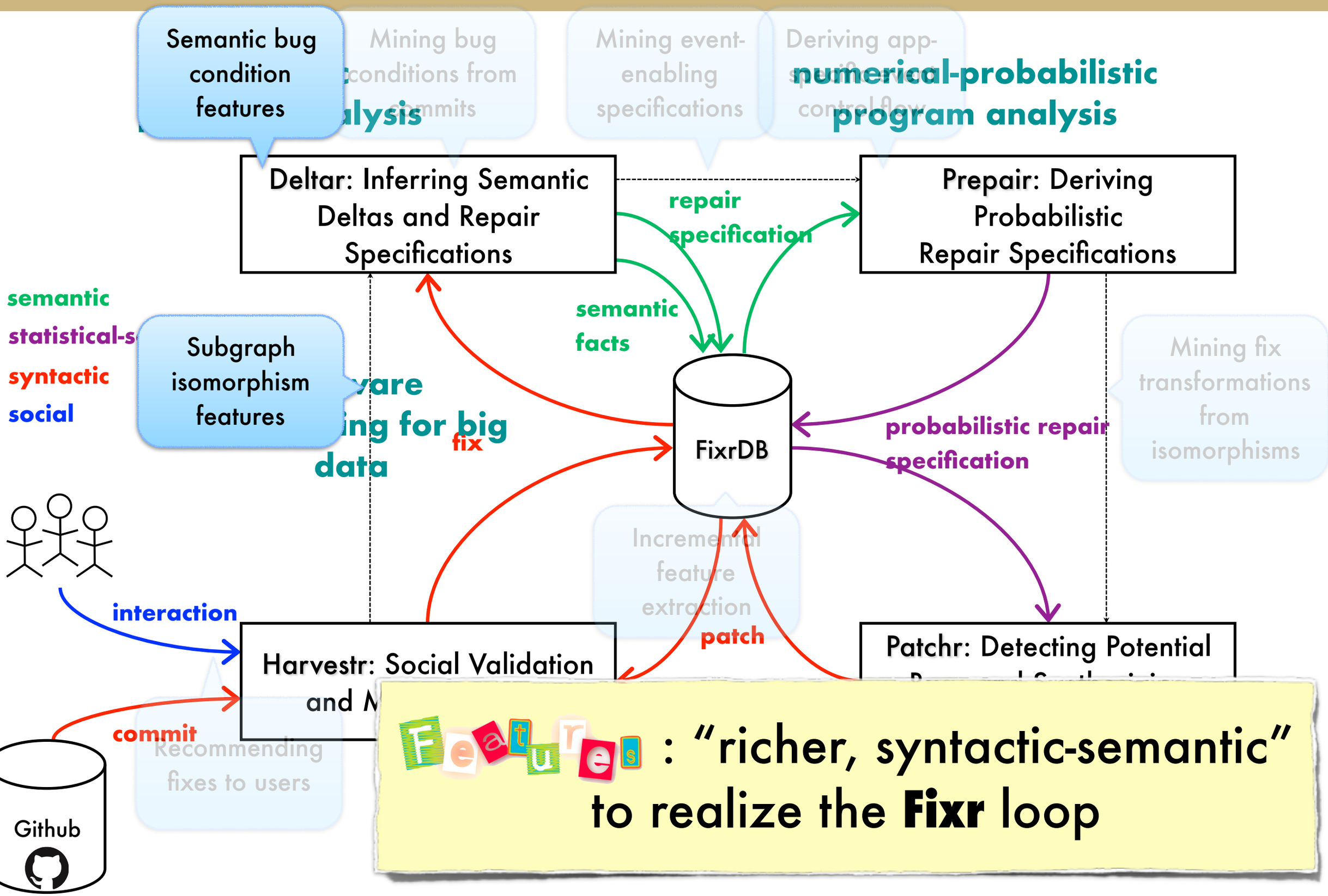
```
db.beginTransaction()
try {
  ...
                                    ()
```

Subgraph isomorphisms better characterize essential syntactic changes





Fixr: Phase 2


Semantic bug condition features
Mining bug conditions from commits
Mining event-enabling specifications
Deriving app-specific control-flow
numerical-probabilistic program analysis
Deltar: Inferring Semantic Deltas and Repair Specifications
repair specification
Prepair: Deriving Probabilistic Repair Specifications
semantic
statistical-s
syntactic
social
Subgraph isomorphism features
semantic facts
ing for big data
fix
FixrDB
probabilistic repair specification
Mining fix transformations from isomorphisms
interaction
Incremental feature extraction
patch
Harvestr: Social Validation and M
Patchr: Detecting Potential
commit
Recommending fixes to users
Github
Features : "richer, syntactic-semantic" to realize the Fixr loop

Extract features that expose relevant semantic changes

Semantic bug condition features

db.beginTransaction()

Need: Infer a bug condition characterizing reachable states derived from the change

db.endT

Removed states where state(db) ≠ ended

ction()

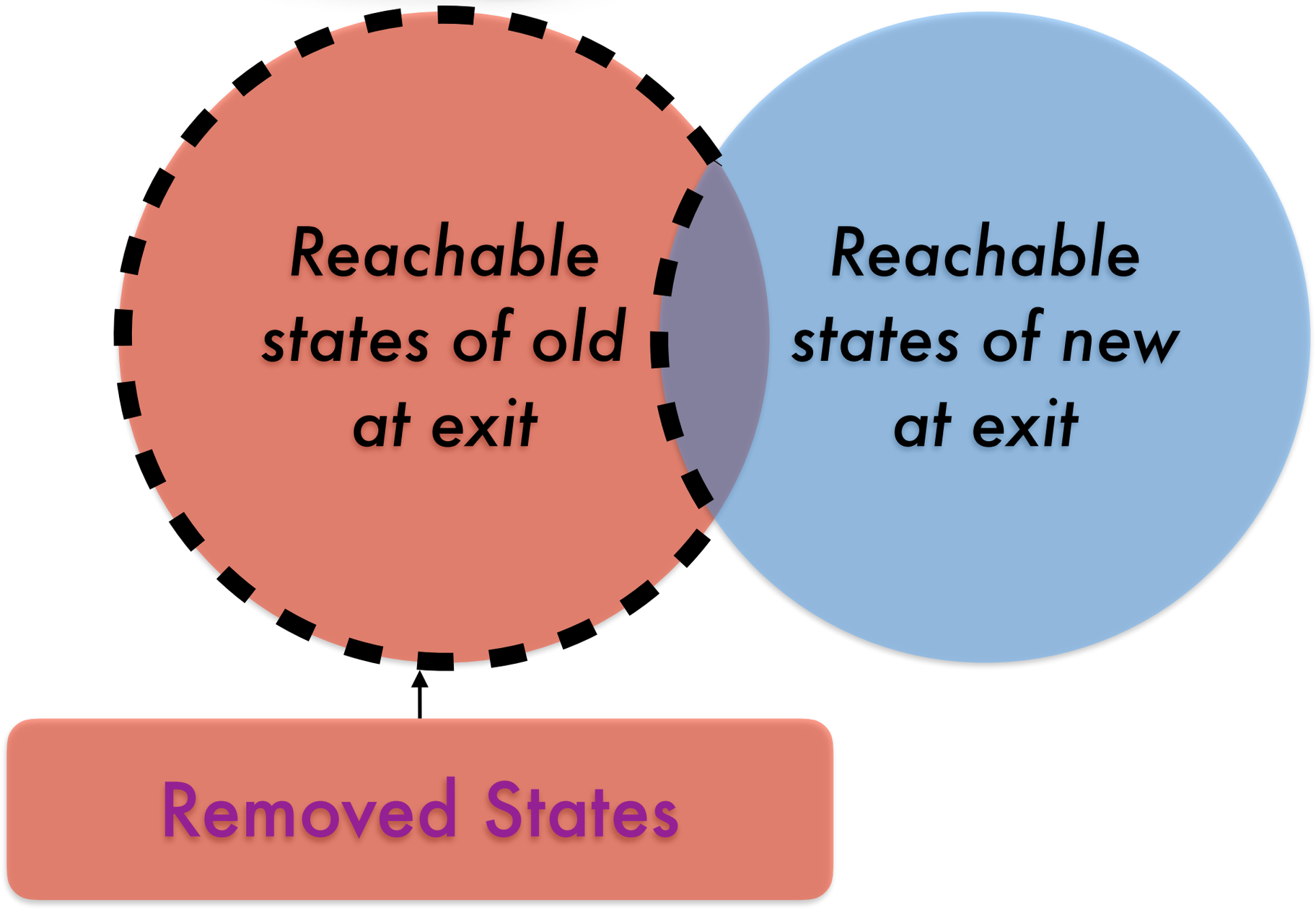
db.endTransaction()

exit

exit

Diff in terms of reachable states

Challenge: Approximate removed states and the relationship to the "underlying" bug condition



Fixr: Phase 2

2016				2017		
Mar	Jun	Sep	Dec	Mar	Jun	Sep
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

12:30-12:45 **Introduction** (Bor-Yuh Evan Chang)

12:45-1:15 **Phase 1 Review** (Sergio Mover)

1:15-1:30 **Phase 2 Preview** (Bor-Yuh Evan Chang)

Richer Feature Extraction

1:30-2:10 Understanding Bug Fixes with Approximate Graph Isomorphisms (Sriram Sankaranarayanan)

2:10-2:30 **Break**

Supporting Richer Feature Extraction

2:30-3:10 Abstracting Event-Driven Systems with Lifestate Rules (Shawn Meier, PhD Student)

3:10-3:30 Active Learning for Lifestate Automata (Krishna Chaitanya Sripada, MS Student)

Interacting with Fixr Users

3:30-3:50 Interactive Decision Support for Transferring Bug-Fixes (Ryo Suzuki, PhD Student)

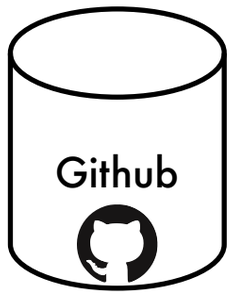
Infrastructure Support for Big Code

3:50-4:10 Incremental Commit/Feature Extraction (Kenneth M. Anderson)

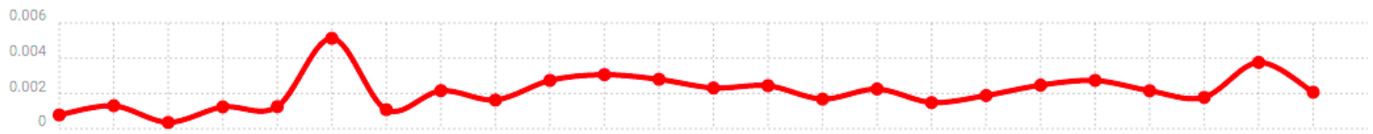
4:10-4:30 **Discussion**

Fixr Highlights

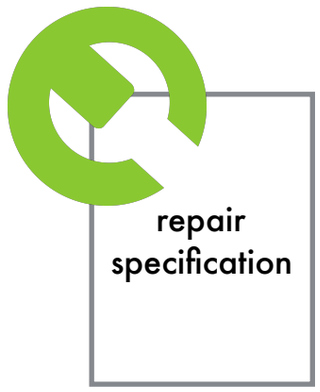
Commits + Time Series = Trends



+



Semantic Repair Specification



Android API Evolution