Fixr: Mining and Understanding Bug Fixes for App-Framework Protocol Defects (TA2)

Bor-Yuh Evan Chang  Ken Anderson  Pavol Cerny  Sriram Sankaranarayanan  Tom Yeh

University of Colorado Boulder

MUSE Kickoff
October 15, 2014
A bug that manifests spectacularly …
A bug that manifests spectacularly …
A bug that manifests spectacularly …
A bug that manifests spectacularly …
A bug that manifests spectacularly ... 

crash

causd by an app-created memory leak
Ask framework devs ...
Ask framework devs ...
Ask framework devs ...

“Do not keep long-lived references to a context-activity”
Ask framework devs ...

“Do not keep long-lived references to a context-activity”

I don’t know how I created a long-lived reference to an Activity!
Ask framework devs ...

“Do not keep long-lived references to a context-activity”

I don’t know how I created a long-lived reference to an Activity!

Often: A misunderstanding of a library causes the library to keep the Activity reference.
Ask framework devs ...

“Do not keep long-lived references to a context-activity”

Often: A misunderstanding of a library causes the library to keep the Activity reference.

Bug from violating (implicit) framework protocol rules
Imagining a post-MUSE scenario ...

for

I don’t know how I created a long-lived reference to an Activity!
Elsewhere, following the state of practice for debugging leaks ...
Elsewhere, following the state of practice for debugging leaks ...  

1. Run the app
Elsewhere, following the state of practice for debugging leaks ...

1. Run the app
2. Watch the heap usage
Elsewhere, following the state of practice for debugging leaks...

1. Run the app
2. Watch the heap usage
3. Dump the heap. Dig around and finally find the culprit!
Elsewhere, following the state of practice for debugging leaks ...

1. Run the app
2. Watch the heap usage
3. Dump the heap. Dig around and finally find the culprit!
Elsewhere, following the state of practice for debugging leaks ...

1. Run the app
2. Watch the heap usage
3. Dump the heap. Dig around and finally find the culprit!
4. Commit a bugfix
Elsewhere, following the state of practice for debugging leaks ...

1. Run the app
2. Watch the heap usage
3. Dump the heap. Dig around and finally find the culprit!
4. Commit a bugfix
5. Bugfix is picked up by Fixr
A **Fixr**-enabled IDE responds ...

I don’t know how I created a long-lived reference to an Activity!
A **Fixr**-enabled IDE responds ...

I don't know how I created a long-lived reference to an Activity!
A **Fixr**-enabled IDE responds ...

It looks like you’ve created a memory leak like and 100,000 others. Would you like to apply ?

I don’t know how I created a long-lived reference to an Activity!
A Fixr-enabled IDE responds ...

It looks like you’ve created a memory leak like and 100,000 others. Would you like to apply ?

the bugfix is “transferred”

I don’t know how I created a long-lived reference to an Activity!
One Sentence Summary: Mine specifications of framework rules (indirectly) from bugfixes

Leverage volume and variety of bugfixes made by the crowd of client app developers
One Sentence Summary: Mine specifications of framework rules (indirectly) from bugfixes

Leverage volume and variety of bugfixes made by the crowd of client app developers
One Sentence Summary: Mine specifications of framework rules (indirectly) from bugfixes

Leverage volume and variety of bugfixes made by the crowd of client app developers

“toolify” stackoverflow
Simple motivating example:
A well-understood Android bug
Simple motivating example: A well-understood Android bug

a common misuse of the framework
Bug (on Android <4)
aView.setTag(..., anObject)
Bug (on Android <4)

aView.setTag(..., anObject)

if anObject can reach aView
Bug (on Android <4)

Framework Invariant

class View {
    static WeakHashMap<View, SparseArray<Object>> sTags;
    Object mTag;
}

createView.setTag(..., anObject)

if anObject can reach aView
class View {
    static WeakHashMap<View, SparseArray<Object>> sTags;
    Object mTag;
}

because of an *unspecified* class invariant: `sTags`’ values (:Object) must not reach their keys (:View)

if anObject can reach aView

Bug (on Android <4)
Bug (on Android <4)

Framework Invariant

A Fix

```java
class View {
    static WeakHashMap<View, SparseArray<Object>> sTags;
    Object mTag;
}
```

because of an unspecified class invariant: sTags’ values (:Object) must not reach their keys (:View)

```
aView.setTag(..., anObject)
aView.setTag(anObject)
```
Bug (on Android <4)

Framework Invariant
because of an unspecified class invariant: `sTags`' values (`Object`) must not reach their keys (`View`)

A Fix
`aView.setTag(..., anObject)` uses `mTag` instead

```
class View {
    static WeakHashMap<View, SparseArray<Object>> sTags;
    Object mTag;
}
```
Bug (on Android <4)

```java
class View {
    static WeakHashMap<View, SparseArray<Object>> sTags;
    Object mTag;
}
```

Because of an unspecified class invariant: `sTags`' values (:Object) must not reach their keys (:View)

Goal: Produce this repair specification: bug pre, framework invariant, fix suggestion
Mining framework specifications with bugfixes

Prior Hypothesis of a Framework Invariant/Rule

Observe a Bugfix

Bayesian Update

Posterior Hypothesis
Mining framework specifications with bugfixes

Prior Hypothesis
of a Framework Invariant/Rule

Observe a Bugfix

Bayesian Update

Posterior Hypothesis

The Fixr Loop:
Create as many observations as possible
The **Fixr** Loop: Component by Component
Fixr: Proposed System
Fixr: Proposed System
**Fixr**: Proposed System

E.g., Two successive versions of source code
Fixr: Proposed System
Fixr: Proposed System

Deltar: Inferring Semantic Deltas and Repair Specifications

MUSE
Fixr: Proposed System

Deltar: Inferring Semantic Deltas and Repair Specifications

MUSE
**Fixr: Proposed System**

Deltar: Inferring Semantic Deltas and Repair Specifications

E.g., Diff in relevant flow-insensitive summary

MUSE

Semantic delta

fix
**Fixr**: Proposed System

Deltar: Inferring Semantic Deltas and Repair Specifications

MUSE
Fixr: Proposed System

Deltar: Inferring Semantic Deltas and Repair Specifications

MUSE

repair specification

semantic delta

fix
Fixr: Proposed System

Deltar: Inferring Semantic Deltas and Repair Specifications

E.g., framework invariant + app "bug pre" + suggestions

MUSE

repair specification

semantic delta

fix
Fixr: Proposed System

Component: **Deltar maps** fixes to semantic difference summaries and candidate repair specifications
Deltar
Deltar

A Fix

aView.setTag(..., anObject)
aView.setTag(anObject)
Deltar

A Fix

Problem: Need to mine and check candidate framework invariants
Deltar

A Fix

Problem: Need to **mine** and **check** candidate framework invariants

Delta

```
WeakHashMap$Entry -> MyView
```

```java
aView.setTag(..., anObject)
aView.setTag(anObject)
```
Deltar

Problem: Need to mine and check candidate framework invariants

A Fix

aView.setTag(..., anObject)

Delta

WeakHashMap$Entry → MyView

Candidate Invariant

sTags == null ∧ mTag != null
Deltar

A Fix

Problem: Need to mine and check candidate framework invariants

Delta

Candidate Invariant

sTags == null \(\land\) mTag != null

Approach: Refine coarse, global summaries and verify candidate invariant on fixed version (scalably with “almost everywhere type analysis”)
Fixr: Proposed System

Deltar: Inferring Semantic Deltas and Repair Specifications

Prepair: Deriving Probabilistic Repair Specifications

E.g., framework invariant + app "bug pre" + fixes

MUSE

repair specification

semantic delta

fix
Fixr: Proposed System

Deltar: Inferring Semantic Deltas and Repair Specifications

Prepair: Deriving Probabilistic Repair Specifications

E.g., framework invariant + app “bug pre” + fixes

MUSE

 repair specification

initial delta

fix
**Fixr: Proposed System**

- **Deltar**: Inferring Semantic Deltas and Repair Specifications
  - **MUSE**
  - **repair specification**
  - **semantic delta**

- **Prepair**: Deriving Probabilistic Repair Specifications
  - **fix**
Fixr: Proposed System

Deltar: Inferring Semantic Deltas and Repair Specifications

MUSE

Prepair: Deriving Probabilistic Repair Specifications

MUSE

fix

repair specification

semantic delta

probabilistic repair specification
**Fixr: Proposed System**

Deltar: Inferring Semantic Deltas and Repair Specifications

Prepair: Deriving Probabilistic Repair Specifications

MUSE

- E.g., generalized repair spec with confidence measure

Fixr: Deriving Probabilistic Repair Specifications
Fixr: Proposed System

Deltar: Inferring Semantic Deltas and Repair Specifications

Prepair: Deriving Probabilistic Repair Specifications

Component: Prepair reduces candidate repair specifications to generalized probabilistic repair specifications
Prepair
Prepair

\[ \text{Candidate Invariant} \]

\[ s\text{Tags} == \text{null} \land m\text{Tag} \neq \text{null} \]
Prepair

sTags == null ∧ mTag != null
∀i. sTags[v][i] ⇒ v
Prepair

Candidate Invariant

\[ s\text{Tags} == \text{null} \land m\text{Tag} \neq \text{null} \]
\[ \forall i. \ s\text{Tags}[v][i] \rightarrow v \]
\[ \forall v. \ s\text{Tags}[v][0] \rightarrow^* v \]
Prepair

**Candidate Invariant**

\[
\text{sTags} == \text{null} \land \text{mTag} \neq \text{null} \\
\forall i. \text{sTags}[v][i] \not\rightarrow v \\
\forall v. \text{sTags}[v][0] \not\rightarrow^* v
\]

**Problem:** Multiple (overly-specific or under-specified) candidate repair specifications
Prepair

Problem: Multiple (overly-specific or under-specified) candidate repair specifications

Candidate Invariant

\[
s\text{Tags} == \text{null} \land m\text{Tag} != \text{null} \\
\forall i. s\text{Tags}[v][i] \rightarrow v \\
\forall v. s\text{Tags}[v][0] \rightarrow^* v
\]

Problem: Multiple (overly-specific or under-specified) candidate repair specifications

Approach: Static analysis as a form of Bayesian updating of priors to derive posteriors. Prevalence of fixes in MUSE database provides priors.
**Prepair**

**Problem:** Multiple (overly-specific or under-specified) candidate repair specifications

**Candidate Invariant**

\[
\text{sTags} == \text{null} \land \text{mTag} != \text{null} \\
\forall i. \text{sTags}[v][i] \rightarrow v \sqcup \\
\forall v. \text{sTags}[v][0] \rightarrow* v
\]

**Approach:** Static analysis as a form of Bayesian updating of priors to derive posteriors. Prevalence of fixes in MUSE database provides priors.
**Fixr**: Proposed System

- **Deltar**: Inferring Semantic Deltas and Repair Specifications
- **MUSE**: Integrating Deltar and Prepair
- **Prepair**: Deriving Probabilistic Repair Specifications
- **Patchr**: Detecting Potential Bugs and Synthesizing Patches

E.g., generalized repair spec with confidence measure
**Fixr**: Proposed System

- **Deltar**: Inferring Semantic Deltas and Repair Specifications
- **Prepair**: Deriving Probabilistic Repair Specifications
- **Patchr**: Detecting Potential Bugs and Synthesizing Patches

E.g., generalized repair spec with confidence measure
**Fixr: Proposed System**

- **Deltar**: Inferring Semantic Deltas and Repair Specifications
- **Prepair**: Deriving Probabilistic Repair Specifications
- **Patchr**: Detecting Potential Bugs and Synthesizing Patches
Fixr: Proposed System

Deltar: Inferring Semantic Deltas and Repair Specifications

Prepair: Deriving Probabilistic Repair Specifications

Patchr: Detecting Potential Bugs and Synthesizing Patches

MUSE

Repair specification

semantic delta

probabilistic repair specification

fix

patch
Fixr: Proposed System

Deltar: Inferring Semantic Deltas and Repair Specifications

Prepair: Deriving Probabilistic Repair Specifications

MUSE

Patchr: Detecting Potential Bugs and Synthesizing Patches

E.g., bug evidence and patch
Fixr: Proposed System

**Deltar:** Inferring Semantic Deltas and Repair Specifications

**Prepair:** Deriving Probabilistic Repair Specifications

**MUSE**

Component: **Patchr**

*maps* (likely buggy) apps to patches

- E.g., bug evidence and patch

**Patchr:** Detecting Potential Bugs and Synthesizing Patches
Patchr
Candidate Invariant

\[ \text{sTags} = \texttt{null} \land \text{mTag} \neq \texttt{null} \]
Problem: How do we validate repair specifications?

Candidate Invariant: \( s\text{Tags} == \text{null} \wedge m\text{Tag} != \text{null} \)

0.9
Problem: How do we validate repair specifications?

Approach: Synthesize patches for human validation (easier to understand and immediately useful)

Candidate
Invariant

\[ s\text{Tags} = \text{null} \land m\text{Tag} \neq \text{null} \]

0.9
**Problem:** How do we validate repair specifications?

**Candidate Invariant:**

\[ sTags = \textbf{null} \land mTag \neq \textbf{null} \]

**Approach:** Synthesize patches for human validation (easier to understand and immediately useful)

**A Patch**

otherView.setTag(..., o)

corrected to:

otherView.setTag(o)
**Problem**: How do we validate repair specifications?

**Candidate Invariant**: \( s\text{Tags} == \textbf{null} \land m\text{Tag} \neq \textbf{null} \)

**Approach**: Synthesize patches for human validation (easier to understand and immediately useful)

**A Patch**: `otherView.setTag(..., o)`

need to find apps satisfying “bug pre”
Fixr: Proposed System

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

MUSE

E.g., bug evidence and patch
Fixr: Proposed System

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

E.g., bug evidence and patch
**Fixr: Proposed System**

- **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

Diagram elements:
- **MUSE**: Central node
- **fix**
- **patch**
- **repair specification**
- **semantic delta**
- **probabilistic repair specification**

Flow arrows indicate the connections and processes within the system.
**Fixr: Proposed System**

- **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

Flow diagram showing the integration of different components:
- MUSE
- **repair specification**
- **semantic delta**
- **fix**
- **social delta**
- **probabilistic repair specification**
- **patch**
**Fixr: Proposed System**

- **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

**Example**: bugfix confirmation

**Diagram Notes**:
- MUSE
- Repair specification
- Semantic delta
- Patch
- Social delta
Fixr: Proposed System

- 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

E.g., bugfix confirmation

interaction

repair specification

semantic delta

social delta

patch

probabilistic repair specification
Fixr: Proposed System

- **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

E.g., bugfix confirmation
Fixr: Proposed System

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

MUSE

repair specification

semantic delta

fix

social delta

patch

probabilistic repair specification
Fixr: Proposed System

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

MUSE
Fixr: Proposed System

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

Code

interaction

commit

MUSE

repair specification

semantic delta

fix

social delta

patch

probabilistic repair specification
**Fixr: Proposed System**

Component: **Harvestr** maps commits and patches to candidate fixes
Harvestr
Harvestr
Problem: How do we find relevant bugfixes?
Problem: How do we find relevant bugfixes?
Problem: How do we find relevant bugfixes?

Approach: Mine meta-data artifacts
**Fixr: Proposed System**

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

Diagram:
- Code → Interaction → Harvestr
- Harvestr → Social Delta → MUSE
- MUSE → Semantic Delta → Deltar
- Deltar → Repair Specification → Prepair
- Prepair → Probabilistic Repair Specification → Patchr
- Patchr → Patch → Harverstr
- Harverstr → Commit → Code
**Fixr: Proposed System**

**Deltar: Inferring Semantic Deltas and Repair Specifications**

**Prepair: Deriving Probabilistic Repair Specifications**

**Goal:** Create a positive feedback loop to derive high-confidence repair specifications
Fixr: Proposed System

Deltar: Inferring Semantic Deltas and Repair Specifications

Prepair: Deriving Probabilistic Repair Specifications

MUSE

Goal: Create a positive feedback loop to derive high-confidence repair specifications
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

Code

semantic
class=\"text\" style=\"position: absolute; color: red;\">semantic
statistical-semantic
class=\"text\" style=\"position: absolute; color: green;\">statistical-semantic
syntactic
class=\"text\" style=\"position: absolute; color: red;\">syntactic
social
class=\"text\" style=\"position: absolute; color: red;\">social

interaction
class=\"text\" style=\"position: absolute; color: blue;\">interaction
commit
class=\"text\" style=\"position: absolute; color: red;\">commit

MUSE

repair specification

fix

semantic delta

social delta

probabilistic repair specification

patch

fix

repair specification
Evaluation Questions
Evaluation Questions

- Iterative and incremental design and evaluation of the Fixr loop
Evaluation Questions

• Iterative and incremental design and evaluation of the **Fixr** loop

• Effectiveness of **Bugfix Transfer**: Given an isolated bugfix, can we derive high-quality repair specifications to lead to useful patches?
Evaluation Questions

- Iterative and incremental design and evaluation of the **Fixr** loop
- Effectiveness of **Bugfix Transfer**: Given an isolated bugfix, can we derive high-quality repair specifications to lead to useful patches?
Evaluation Questions

- Iterative and incremental design and evaluation of the **Fixr** loop

- Effectiveness of **Bugfix Transfer**: Given an isolated bugfix, can we derive high-quality repair specifications to lead to useful patches?

- Effectiveness of **Bugfix Seeding**: Can we isolate likely bugfixes from source repositories?
Evaluation Questions

- Iterative and incremental design and evaluation of the \textbf{Fixr} loop

- Effectiveness of \textbf{Bugfix Transfer}: Given an isolated bugfix, can we derive high-quality repair specifications to lead to useful patches?

- Effectiveness of \textbf{Bugfix Seeding}: Can we isolate likely bugfixes from source repositories?
Prepair for Fixr

www.cs.colorado.edu/∼bec
pl.cs.colorado.edu

PLV
Most Relevant References

**Deltar**


**Prepair**


**Patchr**


**Harvestr**


**Fixr Database**