Understanding and Tooling Framework API Evolution

Wei Wu

Supervised by
Yann-Gaël Guéhéneuc and Giuliano Antoniol
Outlines

- Framework API evolution, problem?
- API change and usage mining
  - Previous works
  - Tooling
  - Dataset
  - General study
  - Detailed study
- API change rule building
  - Previous works
  - Usefulness study
  - Feature usage study
- Conclusion and perspectives
Outlines

- Framework API evolution, problem?
- API change and usage mining
  - Previous works
  - Tooling
  - Dataset
  - General study
  - Detailed study
- API change rule building
  - Previous works
  - Usefulness study
  - Feature usage study
- Conclusion and perspectives
Framework API Evolution?

Framework/Client Program

Client Program

Framework

API

```java
public class SampleHandler extends AbstractHandler {
    /**
     * the command has been executed, so extract the needed information
     * from the application context.
     */
    public Object execute(ExecutionEvent event) throws ExecutionException {
        // Implemented by client programs
        return null;
    }
}
```
Framework API Evolution?

Framework

API

```java
/**
 * An implementation for the extension registry API.
 */
public class ExtensionRegistry implements IExtensionRegistry {
    public void stop() {

    }

    public void stop(Object key) {
}
```
JPMorgan Chase Hacking Affects 76 Million Households

By JESSICA SILVER-GREENBERG, MATTHEW GOLDSTEIN and NICOLE PERLROTH

OCTOBER 2, 2014 12:50 PM

The Manhattan headquarters of JPMorgan Chase, which securities filings revealed was attacked by hackers over the summer. Andrew Burton/Getty Images
Problem!

- Evolving with frameworks is costly
  - Raemaekers et al. (ICSM 2012)
    - Upgrading an authentication framework
    - A whole week of work
  - Linux Debian Distribution
    - Upgrading Perl from 5.10 to 5.12 took
    - Seven weeks to complete
Problem!

- Cost depends on many factors
  - Just considering APIs
    - Which frameworks do we use?
    - Which versions do we use?
    - Which APIs do we use?
    - To which versions do we upgrade?
    - How are the APIs changed?
    - How do we use the APIs?
Problem!

Study API change and usage on a large scale
Outlines

- Framework API evolution, problem?
- API change and usage mining
  - Previous works
  - Tooling
  - Dataset
  - General study
  - Detailed study
- API change rule building
  - Previous works
  - Usefulness study
  - Feature usage study
- Conclusion and perspectives
Previous Works

- API changes
  - Cossette and Walker (2012)
    - API changes in Struts, Log4j
  - Des Rivières (2008)
    - API change classification (about 150 types)
    - No detection
  - Hou and Yao (2011)
    - Reasons for API changes in AWT and Swing

No large-scale study on API changes
Previous Works

API usages

- Businge et al. (2013)
  - Official and internal API usages across the plug-ins
- Lämmel et al. (2011)
  - API usages in SourceForge on a large scale
- Roover et al. (2013)
  - Studies from various angles, such as intent, stakeholders, etc. in enhanced QUALITAS corpus

No studies on API usage and changes
Previous Works

- API usages and changes
  - Robbes et al. (2012)
    - Deprecated APIs in Smalltalk
  - Dietrich et al. (2014)
    - Binary API incompatibilities in QUALITAS corpus
    - Only 8 affected client programs

Studies limited to specific API changes
Solution

- Study API change and usage together on a large scale to answer
  - RQ1: How do framework APIs change?
  - RQ2: How do framework API changes affect client programs?

- Need a tool to collect relevant data
Outlines

- Framework API evolution, problem?
- API change and usage mining
  - Previous works
  - Tooling
  - Dataset
  - General study
  - Detailed study
- API change rule building
  - Previous works
  - Usefulness study
  - Feature usage study
- Conclusion and perspectives
Tooling – ACUA

- API Change and Usage Auditor to collect API-related data
Tooling – ACUA

- Generating API change and usage reports
  - Which frameworks do we use?
  - Which versions do we use?
  - Which APIs do we use?
  - To which versions do we upgrade?
  - How are the APIs changed?
  - How do we use the APIs?
Outlines

- Framework API evolution, problem?
- API change and usage mining
  - Previous works
  - Tooling
  - Dataset
  - General study
  - Detailed study
- API change rule building
  - Previous works
  - Usefulness study
  - Feature usage study
- Conclusion and perspectives
API Change and Usage Mining

14,987 programs evolved for 20 years

Top 11 framework releases with most API changes affecting client programs
Outlines

- Framework API evolution, problem?
- API change and usage mining
  - Previous works
  - Tooling
  - Dataset
  - General study
  - Detailed study
- API change rule building
  - Previous works
  - Usefulness study
  - Feature usage study
- Conclusion and perspectives
RQ1: How do frwk APIs change?

API changes in 59% of frameworks and in 24% of their releases
RQ1: How do frwk APIs change?

At method level, 10% of APIs are changed, only 2% are deprecated.
RQ2: How do frwk API changes affect client programs?

API changes affect 49% of client programs and 21% of their releases.
RQ2: How do frwk API changes affect client programs?

API changes affect only 3% of the APIs used by client programs, none deprecated.
Summary

- **Program-level**
  - Framework API changes happen (59%)
  - Client programs are affected (49%)

- **Method-level**
  - 10% of APIs are changed
  - 3% of used APIs are affected

- **Developers do not document API changes**
  - 2% of the changed APIs are deprecated
  - None of them are used by client programs
Outlines

- Framework API evolution, problem?
- API change and usage mining
  - Previous works
  - Tooling
  - Dataset
  - General study
  - Detailed study
- API change rule building
  - Previous works
  - Usefulness study
  - Feature usage study
- Conclusion and perspectives
Dataset

RQ1: How do frwk APIs evolve?

Missing classes and methods are the most frequent: 46% of total API changes
RQ2: How do frwk API changes affect client programs?

Missing classes and methods affect client programs most frequently (40%)
Summary

- Missing classes and methods
  - 46% of API changes
  - 40% of API changes affect client programs
- Insufficient documentations
- API change rules help developers find the replacements of these missing APIs
Remedy for Missing Classes and Methods

- API change rule
  - A map between a missing API and its replacement in a new release of a framework
  - Target methods represent missing APIs

VCardComposer.shouldAppendCharsetAttribute(List<String>)

VCardBuilder.shouldAppendCharsetParam(String[])

Outlines

- Framework API evolution, problem?
- API change and usage mining
  - Previous works
  - Tooling
  - Dataset
  - General study
  - Detailed study
- API change rule building
  - Previous works
  - Usefulness study
  - Feature usage study
- Conclusion and perspectives
Previous Approaches

- AURA (2010)
- HiMa (2012)
- MadMatch (2013)
- Schäfer et al. (2008)
- SemDiff (2011)
- ...

Limitations

- Generated API change rules are imperfect
  - Precisions vary on different frameworks

- No study on the usefulness of imperfect API change rules
Outlines

- Framework API evolution, problem?
- API change and usage mining
  - Previous works
  - Tooling
  - Dataset
  - General study
  - Detailed study
- API change rule building
  - Previous works
  - Usefulness study
  - Feature usage study
- Conclusion and perspectives
API Change Rule Usefulness

- API change rules are useful
- The more accurate the API change rules, the more helpful

- How can we improve the accuracy of API change rule building?
Outlines

- Framework API evolution, problem?
- API change and usage mining
  - Previous works
  - Tooling
  - Dataset
  - General study
  - Detailed study
- API change rule building
  - Previous works
  - Usefulness study
  - Feature usage study
- Conclusion and perspectives
AURA

<table>
<thead>
<tr>
<th>Approach</th>
<th>Call-Dependence</th>
<th>Signature</th>
<th>Inheritance</th>
<th>Source Comment</th>
<th>Metrics</th>
<th>Structural</th>
<th>Comment</th>
<th>SVN</th>
</tr>
</thead>
<tbody>
<tr>
<td>SemDiff</td>
<td>√</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Beagle</td>
<td>√</td>
<td>√</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>√</td>
</tr>
<tr>
<td>S. Kim et al.</td>
<td>√</td>
<td>√</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M. Kim et al.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MADMatch</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HiMa</td>
<td>√</td>
<td>√</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Schäfer et al.</td>
<td>√</td>
<td>√</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AURA</td>
<td>√</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UMLDiff</td>
<td>√</td>
<td>√</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Combination of Features

- Single feature
  - SemDiff
  - M. Kim et al.

- Prioritised features
  - Explicit: AURA, HiMa, Schäfer et al.
  - Weighting: Beagle, S. Kim et al.
  - Mixed: MADMatch, UMLDiff
Limitations

- No study on the effectiveness of features
- No study on the effectiveness of the combinations of features
- Prioritised multi-feature approaches
  - Potential contradictions among features
    - High priority features shadow lower priority ones
    - Hard to extend with new/different features
AURA

MOOP: multi-objective optimization problem

- Problems with potential conflicting objectives
- Solved by computing Pareto optimal solutions
Research Questions

- RQ1: How effective are the features used in the literature to build API change rules?

- RQ2: Can we use MOOP techniques to improve over prioritised multi-feature approaches?
MOFAE

- MOFAE: multi-objective framework for API evolution
  - Reformulates API change rule building
  - as a MOOP
  - Uses features as objectives
  - Uses jMetal MOOP algorithm framework
  - Allows flexible feature configuration

## Four Experiment Features

<table>
<thead>
<tr>
<th>Approach</th>
<th>Call-Dependency</th>
<th>Signature</th>
<th>Inheritance</th>
<th>Source Comment</th>
<th>Metrics</th>
<th>Structural</th>
<th>Comment</th>
<th>SVN</th>
</tr>
</thead>
<tbody>
<tr>
<td>SemDiff</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Beagle</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>□</td>
<td></td>
<td>□</td>
<td></td>
</tr>
<tr>
<td>S. Kim et al.</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>□</td>
<td></td>
<td>□</td>
<td></td>
</tr>
<tr>
<td>M. Kim et al.</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MADMatch</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>□</td>
<td></td>
<td>□</td>
<td></td>
</tr>
<tr>
<td>HiMa</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>□</td>
</tr>
<tr>
<td>Schäfer et al.</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AURA</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UMLDiff</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>
Existing Approaches

- Difficulties to compare
  - Not all available
  - Not always executable
  - One recommendation per target method
Experiment Approaches

- Single-feature approaches
- Multi-feature approaches

<table>
<thead>
<tr>
<th>Prioritised</th>
<th>MOFAE</th>
<th>Feature Combination</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>M1</td>
<td>Call-dependency + Signature</td>
</tr>
<tr>
<td>P2</td>
<td>M2</td>
<td>Source code comments + Signature</td>
</tr>
<tr>
<td>P3</td>
<td>M3</td>
<td>Inheritance + Signature</td>
</tr>
<tr>
<td>PA</td>
<td>MA</td>
<td>Call-dependency + Inheritance + Source code comments + Signature</td>
</tr>
</tbody>
</table>
Experiment Approaches

- Outputs
  - MOFAE
    - Maximum 6 recommendations
  - Single-feature and prioritized approaches
    - Top 6 recommendations

- More conservative for MOFAE approaches
Experiment Approaches

Comparison

- Number of target methods with correct replacements
- Correct recommendation position

```java
MeterPlot.getDialBorderColor()

MeterPlot.getDialBackgroundPaint()
MeterPlot.getDialOutlinePaint()
MeterPlot.getNormalPaint()
MeterPlot.getCriticalPaint()
MeterPlot.getValueFont()
 MeterPlot.getNeedlePaint()
```
## Target Frameworks

<table>
<thead>
<tr>
<th>Framework</th>
<th>Releases</th>
<th># Methods</th>
<th># Target Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>Android SDK</td>
<td>2.1_r2.1p2</td>
<td>20,516</td>
<td>106</td>
</tr>
<tr>
<td></td>
<td>2.2.3_r2</td>
<td>21,214</td>
<td></td>
</tr>
<tr>
<td>jEdit</td>
<td>4.1</td>
<td>2,774</td>
<td>87</td>
</tr>
<tr>
<td></td>
<td>4.2</td>
<td>3,547</td>
<td></td>
</tr>
<tr>
<td>jFreeChart</td>
<td>0.9.11</td>
<td>4,751</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>0.9.12</td>
<td>5,197</td>
<td></td>
</tr>
<tr>
<td>jHotDraw</td>
<td>5.2</td>
<td>1,486</td>
<td>43</td>
</tr>
<tr>
<td></td>
<td>5.3</td>
<td>2,265</td>
<td></td>
</tr>
<tr>
<td>Log4j</td>
<td>1.0.4</td>
<td>906</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>1.1.3</td>
<td>1,110</td>
<td></td>
</tr>
<tr>
<td>Struts</td>
<td>1.1</td>
<td>5,973</td>
<td>91</td>
</tr>
<tr>
<td></td>
<td>1.2.4</td>
<td>6,111</td>
<td></td>
</tr>
</tbody>
</table>
RQ1: How effective are the features to identify change rules?

<table>
<thead>
<tr>
<th># Correct</th>
<th>Call-dependency</th>
<th>Inheritance</th>
<th>Comment</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Android SDK</td>
<td>22</td>
<td>9</td>
<td>20</td>
<td>50/106</td>
</tr>
<tr>
<td>jEdit</td>
<td>12</td>
<td>5</td>
<td></td>
<td>30/87</td>
</tr>
<tr>
<td>jFreeChart</td>
<td>5</td>
<td></td>
<td>15</td>
<td>29/30</td>
</tr>
<tr>
<td>jHotDraw</td>
<td>1</td>
<td>14</td>
<td>20</td>
<td>36/43</td>
</tr>
<tr>
<td>Log4j</td>
<td>2</td>
<td>3</td>
<td>8</td>
<td>12/15</td>
</tr>
<tr>
<td>Struts</td>
<td>2</td>
<td>17</td>
<td>8</td>
<td>19/91</td>
</tr>
</tbody>
</table>

The smaller, the better

Signatures are most effective
RQ2: Can we use MOOP techniques to improve multi-feature approaches?

MOFAE: 13% more correct replacements, 3% higher in position
RQ2: Can we use MOOP techniques to improve multi-feature approaches?

MOFAE: 20% more correct replacements
Summary

- Signature is the most effective feature
- MOFAE builds 13% more correct change rules, 3% higher in position
- MOFAE builds 20% correct change rules than MADMatch
Outlines

- Framework API evolution, problem?
- API change and usage mining
  - Previous works
  - Tooling
  - Dataset
  - General study
  - Detailed study
- API change rule building
  - Previous works
  - Usefulness study
  - Feature usage study
- Conclusion and perspectives
Thesis

Following analyses of the reality of API changes and usages, of the usefulness of API change rules, and of the effectiveness of the features used to build these rules, we can build more effective and extensible API change-rule recommendation tools with MOFAE.
Perspectives

- **Near term**
  - Extensive qualitative analyses
  - More effective features
  - Tools for other upgrading tasks
  - Developers’ interviews

- **Long term**
  - Language-supported API visibility
  - Framework API standards
  - Independent framework evaluation
Tooling – ACUA

- API Change and Usage Auditor to collect API-related data

RQ2: How do frwk API changes affect client programs?

- Missing classes and methods affect client programs most frequently (40%)

API Chang Rule Usefulness

RQ2: Can we use MOOP techniques to improve multi-feature approaches?

MOFAE: 13% more correct replacements, 3% higher in position