Extending Existing Inference Tools to Mine Dynamic APIs

Ziyad Alsaeed and Michal Young
University of Oregon
June 2, 2018
2nd International Workshop on API Usage and Evolution (WAPI)
Motivation

• API understanding is a key to solve many software usage issues.

• Software documentations are rarely up-to-date and constraints associated with objects are usually in the brain of the creator.

• How do we capture the software dynamic nature.
Existing Dynamic Inference Tools (Daikon)

The Majority of our contribution extremely tied to established

Original program Instrumented program
Instrument Run Data trace database
Chicory

Detect invariants
Daikon

Test suite

[1] Ernst et. al. ICSE’99
Mining the Well-Known StackAr by Daikon

Constructor:
StackAr (int capacity)

Public methods:
void push(x)
void pop()
Object top()
Object topAndPop()
boolean isEmpty()
boolean isFull()
void makeEmpty()
How Chicory Works

Legend

- User-Defined Field
- Primitive

$\text{max}\_\text{depth} = 2$

$\text{Method } x \rightarrow \text{EXIT}$

$\text{PassedArguments}$

$\text{ReturnValue}$

$\text{Fields}$
How Chicory Works

<table>
<thead>
<tr>
<th>x:::EXIT</th>
<th>$f_{11}$</th>
<th>$f_{121}$</th>
<th>...</th>
<th>$a_{n11}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>invocation</td>
<td>1</td>
<td>null</td>
<td></td>
<td>0.0</td>
</tr>
</tbody>
</table>

Legend:
- User Defined Field
- Primitive

$max_{\text{depth}} = 2$
How Chicory Works

<table>
<thead>
<tr>
<th>x:::EXIT</th>
<th>$f_{11}$</th>
<th>$f_{121}$</th>
<th>$a_{n11}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>invocation$_1$</td>
<td>1</td>
<td>null</td>
<td>0.0</td>
</tr>
<tr>
<td>invocation$_2$</td>
<td>-1</td>
<td>null</td>
<td>0.0</td>
</tr>
</tbody>
</table>
How Chicory Works

<table>
<thead>
<tr>
<th>x:::EXIT</th>
<th>$f_{11}$</th>
<th>$f_{121}$</th>
<th>$a_{n11}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>invocation</td>
<td>1</td>
<td>null</td>
<td>0.0</td>
</tr>
<tr>
<td>invocation</td>
<td>-1</td>
<td>null</td>
<td>0.0</td>
</tr>
<tr>
<td>invocation</td>
<td>-2</td>
<td>null</td>
<td>0.0</td>
</tr>
</tbody>
</table>

```
null
6/2/2018 WAPI'18 - Z. Alsaeed & M. Young
```
How Chicory Works

<table>
<thead>
<tr>
<th>invocation _1</th>
<th>( f_{11} )</th>
<th>( f_{121} )</th>
<th>( a_{n11} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>invocation _1</td>
<td>1</td>
<td>null</td>
<td>0.0</td>
</tr>
<tr>
<td>invocation _2</td>
<td>-1</td>
<td>null</td>
<td>0.0</td>
</tr>
<tr>
<td>invocation _3</td>
<td>-2</td>
<td>null</td>
<td>0.0</td>
</tr>
</tbody>
</table>

\( \text{invocation}_n \) | \( \theta \) | null | 0.0

...
What if?
Simplified Real-World Example

```java
public class Modifier {
    public List<Receiver> receivers = new ArrayList<Receiver>();

    public void addReceiver (Receiver rcv) {
        receivers.add(rcv);
    }

    public void modify () {
        for (Receiver rcv: receivers)
            rcv.increment();
    }
}
```
Simplified Real-World Example

```java
1 public class Receiver {
2   public int internalValue = 0;
3
4   public void increment() {
5       internalValue += 1;
6   }
7 }
```
Variables Structure of *modify* based on Chicory:

```
Method modify() \rightarrow ENTER

Fields

Arguments

receivers

max_depth = 2
```

Legend

- User-Defined Field
- Primitive
- Complex Field
Possible Scenarios When Observing a Dynamic Data Structure

• Element introduction (added to DDS):
  • An element that was not present in the DDS until a later point in the program execution and never removed thereafter.

• Element removal (removed from DDS):
  • An element that exists at some point of the program execution but removed before the last observation of the program point.

• And more ...
Possible Structure Evolution

![Diagram of possible structure evolution]

Legend:
- User-Defined Field
- Primitive
- Complex Field

Time:
- \(\text{invocation}_1\)
- \(\text{invocation}_2\)
- \(\ldots\)
- \(\text{invocation}_n\)

Fields
- \(\text{modify}() \rightarrow \text{ENTER}\)

Arguments
- \(\text{modify}() \rightarrow \text{ENTER}\)

Max depth = 2

- \(\text{receiver}_1\)
- \(\text{receiver}_2\)
- \(\ldots\)
- \(\text{receiver}_n\)

Internal Value

6/2/2018
WAPI'18 - Z. Alsaeed & M. Young
Relation to Daikon

• Daikon expects a very well defined structure of a program point (method entrance or exit).

• Only one variable structure tree per program point.
eChicory Structure

Target Application

Chicory

Structure and Traces Tracker

Structure and Traces Unifier

eChicory

Daikon (version released Jan 5, 2017)

Invariants Based on Chicory

Invariants Based on eChicory

Unification Phase in eChicory (element introduction)
Unification Phase in eChicory (element introduction)

Unified Version

Method modify() → ENTER

Fields

Arguments

receivers

receivers

receivers

modify:::ENTER

<table>
<thead>
<tr>
<th></th>
<th>$r_1$.internalValue</th>
<th>$r_2$.internalValue</th>
<th>$r_3$.internalValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>$invocation_1$</td>
<td>0</td>
<td>nonsensical</td>
<td>nonsensical</td>
</tr>
<tr>
<td>$invocation_2$</td>
<td>1</td>
<td>0</td>
<td>nonsensical</td>
</tr>
<tr>
<td>$invocation_3$</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Unification Phase in eChicory (element removal)

• Given Daikon design, removing a variable can only be achieved by manipulating its trace.

• Changing a trace to nonsensical after it was initialized is prohibited.

• Given it arbitrary value (e.g. stretch its last known value to the rest of instances where it was removed thereafter) would interfere with the invariants integrity.
Evaluation - Artifact Selection Criteria

• Source available in GitHub.

• Applications of size between 2K and 10K LOC.

• Has indications that one of the selected patterns is used (given the repository issues tracker, pull requests, and wiki).

• Has high test coverage (if reported).

• Popular or well maintained applications based on the star rate or managing organization.
## Evaluation - Selected Artifacts

<table>
<thead>
<tr>
<th>Application</th>
<th>Description</th>
<th>Selected Classes</th>
<th># of Methods</th>
<th>Represented Design Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mockito</td>
<td>Mocking framework for unit tests in Java</td>
<td>InvocationNotifierHandler</td>
<td>7</td>
<td>Observer Pattern</td>
</tr>
<tr>
<td>Apache Struts</td>
<td>Framework for creating Java web applications</td>
<td>DefaultActionInvocation</td>
<td>29</td>
<td>MVC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DefaultUnknownHandlerManager</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>CongurationManager</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>VelocityManager</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>SimpleTextNode</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>SimpleAdapterDocument</td>
<td>43</td>
<td></td>
</tr>
<tr>
<td>JabRef</td>
<td>BibTeX Management application</td>
<td>EntryEditor</td>
<td>22</td>
<td>MVC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CleanupActionsListModel</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>UndoableModifySubtree</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>ImportInspectionDialog</td>
<td>21</td>
<td></td>
</tr>
<tr>
<td>Zeppelin</td>
<td>A web based interactive data analytic tool</td>
<td>Folder</td>
<td>23</td>
<td>Observer Pattern</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Notebook</td>
<td>45</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>NotebookRepoSync</td>
<td>31</td>
<td></td>
</tr>
</tbody>
</table>
Evaluation Criteria (Precision and Recall)

• Pros:
  • Has being the base for evaluating specification miners.
  • Shows a good insight about the accuracy of the specification miner.

• Cons:
  • A ground truth about the test subject must be defined ahead (this is done by humans, thus can’t be scaled).
  • Human defined ground truth, can differed based on the developers view or opinion.
Evaluation Criteria (Purity Analysis)

• The notion of pure (side-effect free) methods is well-defined in the static analysis domain.

• Can be generated automatically and scale with large applications.

• Not the goal of dynamic analysis, but can be used to check consistency.
Mockito - InvocationNotifierHandler

<table>
<thead>
<tr>
<th>Method</th>
<th>jPure</th>
<th>eChicory</th>
<th>Chicory</th>
</tr>
</thead>
<tbody>
<tr>
<td>InvocationNotifierHandler(InternalMockHandler&lt;T&gt;, MockCreationSettings&lt;T&gt;)</td>
<td>!pure</td>
<td>!pure</td>
<td>!pure</td>
</tr>
<tr>
<td>handle(Invocation)</td>
<td>!pure</td>
<td>!pure</td>
<td>pure</td>
</tr>
<tr>
<td>notifyMethodCall(Invocation, Object)</td>
<td>!pure</td>
<td>!pure</td>
<td>pure</td>
</tr>
<tr>
<td>notifyMethodCallException(Invocation, Throwable)</td>
<td>!pure</td>
<td>!pure</td>
<td>pure</td>
</tr>
<tr>
<td>getMockSettings()</td>
<td>!pure</td>
<td>pure</td>
<td>pure</td>
</tr>
<tr>
<td>getInvocationContainer()</td>
<td>!pure</td>
<td>pure</td>
<td>pure</td>
</tr>
<tr>
<td>setAnswersForStubbing(List&lt;Answer&lt;?&gt;&gt;&gt;)</td>
<td>!pure</td>
<td>pure</td>
<td>pure</td>
</tr>
</tbody>
</table>

**Total number of reported methods with no indication of effect**

<table>
<thead>
<tr>
<th></th>
<th>jPure</th>
<th>eChicory</th>
<th>Chicory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td>0</td>
<td>3</td>
<td>6</td>
</tr>
</tbody>
</table>
Mockito - InvocationNotifierHandler

Chicory

```
124
125 org.mockito.internal.handler.InvocationNotifierHandler.handle(org.mockito.invocation.Invocation)::{ENTER
126
127 invocation != null
128 invocation.getClass().getName() ==
129 org.mockito.internal.creation.bytebuddy.InterceptedInvocation.class
130 this.invocationListeners.getClass().getName() !=
131 orig(this.invocationListeners).getClass().getName() !=
132 orig(this.invocationListeners).getClass().getName() !=
133 orig(this.invocationListeners).getClass().getName() !=
134 orig(this.mockHandler.getClass().getName()) !=
135 return getClass().getName() !=
136 orig(this.invocationListeners.getClass().getName()) !=
137 orig(this.invocationListeners.getClass().getName()) !=
138 orig(this.mockHandler.getClass().getName()) !=
139 orig(this.mockHandler.getClass().getName()) !=
```

eChicory

```
292
293 org.mockito.internal.handler.InvocationNotifierHandler.handle(org.mockito.invocation.Invocation)::{EXIT
294
295 ...
296 ...
```

6/2/2018
Impediments to Observe Other Selected Test Subjects

- Inadequate inputs (unit tests).

- Naive implementation of concrete classes.

- Absence of elements in DDS.

- Tests Failures (confirmed by repository maintainers).
Wrote Comprehensive Tests to Prove our Approach Potential

• To prove the provided tests are the cause of result limitations, we wrote unite tests for one of the classes from Apache Struts.

• Class DefaultUnknownHandlerManager
  • Method-1: handleUnknownAction
  • Method-2: handleUnknownMethod

• Written tests are reviewed and merged into Apache Struts’ main repository.
Performance - Traces Collection Phase
Performance - Inference Phase

<table>
<thead>
<tr>
<th>Task</th>
<th>eChicory Time (Seconds)</th>
<th>Chicory Time (Seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEFAULT_ACTION_INVOCATION</td>
<td>5.32</td>
<td>6.40</td>
</tr>
<tr>
<td>DEFAULT_UNKNOWN_HANDLER</td>
<td>3.72</td>
<td>3.30</td>
</tr>
<tr>
<td>VELOCITY_MANAGER</td>
<td>1.50</td>
<td>1.59</td>
</tr>
<tr>
<td>SIMPLE_TEXT_NODE</td>
<td>2.90</td>
<td>2.79</td>
</tr>
<tr>
<td>SIMPLE_ADAPTER_DOCUMENT</td>
<td>3.31</td>
<td>3.83</td>
</tr>
<tr>
<td>FOLDER</td>
<td>41.62</td>
<td></td>
</tr>
<tr>
<td>NOTEBOOK_REPO_SYNC</td>
<td>13.24</td>
<td>8.69</td>
</tr>
<tr>
<td>INVOCATION_NOTIFY_HANDLER</td>
<td>5.09</td>
<td>3.90</td>
</tr>
</tbody>
</table>
Conclusion

• We highlighted the non-fully dynamic tracing issue and clearly identified that limitations of current instrumentation methodologies.

• We implemented a prototype as a proof of concept to fully dynamically observe complex systems.

• We showed by real world example that existing instrumentation techniques are blind to common design patterns are.

• DDSs are only one source of program structural change. There are different programming practices that leads to very dynamic structure needs to be addressed.
Thank you.

References: