Detecting Inefficient API Usage

David Kawrykow and Martin Robillard
McGill University
Application Programming Interfaces
Inefficient API Usage

gettingURLResponse(String url)
{
    WebResponse response = null;
    URL serverUrl = new URL(url);
    conversation = new WebConversation();
    request = new GetMethodWebRequest(serverUrl,"");
    response = conversation.getResponse(request);
    return response;
}
gettingURLResponse(String url) {
    WebResponse response = null;
    URL serverUrl = new URL(url);
    conversation = new WebConversation();
    request = new GetMethodWebRequest(serverUrl, "");
    response = conversation.getResponse(request);
    return response;
}

goingingURLResponse(String url) {
    return new WebConversation().getResponse(url);
}
Inefficient API Usage

Client Method

Diagram showing inefficient API usage with clients A, B, C, and D.
Inefficient API Usage
Motivation

Quality of Source Code vs Time
Detection is not Easy

```java
getURLResponse(String url)
{
    WebResponse response = null;
    URL serverUrl = new URL(url);
    conversation = new WebConversation();
    request = new GetMethodWebRequest(serverUrl, "");
    response = conversation.getResponse(request);
    return response;
}

getResponse(String url)
{
    return _mainWindow.getResponse(url);
}
```
Our Approach

Abstract Method Bodies

```java
gettingURLResponse(String url) {
    WebResponse response = null;
    URL serverUrl = new URL(url);
    con = new WebConversation();
    req = new GMWR(serverUrl,"");
    response = con.getResponse(req);
    return response;
}
```

- WebResponse
- URL
- URL(String)
- WebConversation()
- GMWR(URL,String)
- getResponse(GMWR)
Our Approach

Abstract Method Bodies

goingtURLResponse(String url)
{
    WebResponse response = null;
    URL serverUrl = new URL(url);
    con = new WebConversation();
    req = new GMWR(serverUrl,"");
    response = con.getResponse(req);
    return response;
}
Our Approach

Find Imitations

Element Reference Match

Client Method

Library Method
Three Kinds of Matches

Direct

Indirect

Nested
Three Kinds of Matches

- Direct
- Indirect
- Nested
Three Kinds of Matches

Direct

Indirect

Nested

Client Method

Other Method

Library Method

Direct Match

Indirect Match
Our Approach

Filter the Imitations

Client Method

Library Method

WebResponse
Our Approach

Abstract Imitations as Usage Patterns

- Direct Match
- Indirect Match
- Nested Match

Imitated Method

A
B
C
Our Approach

Store and Reuse (good) Patterns

```java
gethingURLResponse(String url) {
    WebResponse resp = null;
    ....
    return resp.getResponse(...);
}

gethingURLResponse(String url) {
    WebResponse resp = null;
    ....
    return resp.getResponse(...);
}
```
## Preliminary Results

<table>
<thead>
<tr>
<th>Project</th>
<th>Valid Imitations</th>
<th>Reusable Patterns</th>
<th>“Edits”</th>
</tr>
</thead>
<tbody>
<tr>
<td>JBoss</td>
<td>163</td>
<td>29</td>
<td>209</td>
</tr>
<tr>
<td>SpringFramework</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>JasperServer</td>
<td>13</td>
<td>9</td>
<td>42</td>
</tr>
</tbody>
</table>

**Edits:** Saving a function call, variable declaration, or String / integer literal.
UploadFileSpec[] files = new UploadFileSpec[] 
    { new UploadFileSpec(file) }; 
foo.setParameter(“newData”, files);

Vs.

foo.setParameter(“newData”, file);
Thank You

Questions?