Evolution-Aware Monitoring-Oriented Programming (eMOP)

Owolabi Legunsen, Darko Marinov, and Grigore Roșu

ICSE 2015 (NIER Track)
Florence, Italy
May 21, 2015
Monitoring-Oriented Programming (MOP)

Runtime monitoring of software against formal properties

- **Existing technique** targeted at single program version

**Problems:** High overhead and too many violations shown during evolution across many versions
Evolution-Aware MOP (eMOP)

Make MOP faster and show fewer violations during evolution

- Proposed

![Diagram showing the process of Evolution-Aware MOP with nodes for Code Changes, Property, Tests, MOP, Runtime Monitors, and Violation with an asterisk.](image-url)
Input: (Potentially Buggy) Code

```java
public boolean m(List a, List b) {
    ... 
    for (Iterator i = a.iterator(); i.hasNext();){
        ... 
        for (Iterator i2 = b.iterator(); i.hasNext();){
            ... i2.next() ...
        }
    }
    return ... 
}
```

Line 5 should be `i2.hasNext()`

Mimics two real bugs found in older AspectJ code
Input: Formally Specified Properties

1. When to fire Events
   after `Iterator.hasNext() == true`, before `Iterator.next()`

2. Specification over Events
   `Iterator.hasNext() == true` precedes every `Iterator.next()`

3. Handler code
   User-defined action when specification is violated

Many properties can be monitored at once
public boolean find(List a, List b) {
  ...
  for(Iterator i = a.iterator(); i.hasNext();){
    ...
    for(Iterator i2 = b.iterator(); i2.hasNext();){
      // event: “before Iterator.next()”
      ... i2.next() ...
    }
  }
  return ...
}
Current State of MOP Research

• Many papers, focus on reducing runtime overhead
• Many bugs found in well-used, well-tested code
• **All prior research focused on one version**
  • Recurring costs of monitoring are high, e.g.,

<table>
<thead>
<tr>
<th>Run</th>
<th>Properties Monitored</th>
<th>Total Violations</th>
<th>Time(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>No MOP v1</td>
<td>n/a</td>
<td>n/a</td>
<td>8.4</td>
</tr>
<tr>
<td>MOP v1</td>
<td>180</td>
<td>27,895</td>
<td>164.1</td>
</tr>
<tr>
<td>MOP v2</td>
<td>180</td>
<td>27,904</td>
<td>231.8</td>
</tr>
</tbody>
</table>
Evolution-Aware MOP (eMOP)

• Improve MOP during software evolution
  • Faster: re-monitor based on parts affected by changes
  • Show fewer violations: show only violations due to changes

• We propose three techniques
  • Can be used separately or combined
    • Property selection
    • Monitor selection
    • Test selection
Technique: Property Selection

- What subset of properties to re-monitor in new version?
- Preliminary evaluation by seeding `i2.next()` bug:
  - Only `Iterator_HasNext` is affected by changes

<table>
<thead>
<tr>
<th>Run</th>
<th>Properties Monitored</th>
<th>Properties Violated</th>
<th>HasNext Violations</th>
<th>Total Violations</th>
<th>Time(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>No MOP v1</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>8.4</td>
</tr>
<tr>
<td>MOP v1</td>
<td>180</td>
<td>6</td>
<td>0</td>
<td>27,895</td>
<td>164.1</td>
</tr>
<tr>
<td>MOP v2</td>
<td>180</td>
<td>7</td>
<td>9</td>
<td>27,904</td>
<td>231.8</td>
</tr>
<tr>
<td>eMOP v2</td>
<td>1</td>
<td>1</td>
<td>9</td>
<td>9</td>
<td>8.8</td>
</tr>
</tbody>
</table>
Technique: Monitor Selection

• Generate monitors for parts of code affected by change
• Example: \texttt{Foo.java} and \texttt{Bar.java} both use Iterator

Old Version

\begin{itemize}
  \item \texttt{Foo.java}
  \item \texttt{Bar.java}
\end{itemize}

New Version

\begin{itemize}
  \item \texttt{Foo.java}
  \item \texttt{Bar.java}
\end{itemize}

Do not generate \texttt{Iterator\_HasNext} Monitors

Generate \texttt{Iterator\_HasNext} Monitors
Technique: Test Selection (MOP + RTS)

• In eMOP we monitor execution of tests
  • RTS selects **subset** of tests that can be affected by code changes
  • If fewer tests are run, fewer violations and less overhead
Some Challenges

• Safely determining properties/monitors/tests that can’t have new violations
• Non-determinism, e.g.,

In these versions, the same tests are run, but different number of violations

(a) Violation Counts for one project
Conclusions

- All prior research on MOP targeted single code versions
- **eMOP** aims to adapt MOP to software evolution
  - Make MOP **faster** between versions of software
  - **Show only violations** due to changes between versions
- We proposed three techniques for eMOP
  - **Property selection**
  - **Monitor selection**
  - **Test selection**
Backup Slides
How MOP Works

Run Time

Compile Time

Code

Property

```java
public boolean findFiles(List files, List dirs){
    File file, dir; int count = 0;
    for(Iterator iter = files.iterator(); iter.hasNext();){
        file = (File) iter.next();
        for(Iterator iter2=dirs.iterator(); iter2.hasNext();){
            dir = (File) iter2.next();
            if (new File(dir, file.getName()).exists()){
                count++; break;  // file is in dir
            }
        }
    }
    return count == files.size(); }

1Iterator_HasNext(Iterator i) {
2    event hasnexttrue after(Iterator i) returning(boolean b);
3    call(*Iterator+.hasNext())&&target(i)&&condition(b)[]
4    event next before(Iterator i);
5    call(*Iterator+.next() && target(i)[]
6    ltl: [](next => (*) hasnexttrue)
7    @violation [...]}
8}
```
MOP Costs can be quite high

Run Time

Compile Time

Code

Properties
<table>
<thead>
<tr>
<th>No.</th>
<th>Project</th>
<th>LOC</th>
<th>Tests</th>
<th>Violations</th>
<th>BaseTime(s)</th>
<th>MOPTime(s)</th>
<th>Overhead(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>FXForm2</td>
<td>5355</td>
<td>34</td>
<td>1008</td>
<td>5.2</td>
<td>20.9</td>
<td>304.1</td>
</tr>
<tr>
<td>2</td>
<td>JAQ-InABox</td>
<td>3570</td>
<td>1</td>
<td>1</td>
<td>5.1</td>
<td>5.5</td>
<td>7.4</td>
</tr>
<tr>
<td>3</td>
<td>JSqlParser</td>
<td>7786</td>
<td>232</td>
<td>27895</td>
<td>8.9</td>
<td>342.2</td>
<td>3733.2</td>
</tr>
<tr>
<td>4</td>
<td>ObjectLayout</td>
<td>1305</td>
<td>19</td>
<td>0</td>
<td>9.0</td>
<td>22.4</td>
<td>148.8</td>
</tr>
<tr>
<td>5</td>
<td>androlog</td>
<td>2532</td>
<td>8</td>
<td>19</td>
<td>4.0</td>
<td>5.6</td>
<td>41.1</td>
</tr>
<tr>
<td>6</td>
<td>apache.commons-lang</td>
<td>63425</td>
<td>2497</td>
<td>0</td>
<td>23.8</td>
<td>29.9</td>
<td>25.4</td>
</tr>
<tr>
<td>7</td>
<td>asterisk-java</td>
<td>35659</td>
<td>215</td>
<td>145</td>
<td>10.5</td>
<td>17.5</td>
<td>65.9</td>
</tr>
<tr>
<td>8</td>
<td>bcel</td>
<td>35827</td>
<td>73</td>
<td>27</td>
<td>7.2</td>
<td>12.2</td>
<td>70.9</td>
</tr>
<tr>
<td>9</td>
<td>commons-beans</td>
<td>33007</td>
<td>1269</td>
<td>0</td>
<td>43.4</td>
<td>962.6</td>
<td>2120.1</td>
</tr>
<tr>
<td>10</td>
<td>commons-cli</td>
<td>6292</td>
<td>364</td>
<td>0</td>
<td>4.7</td>
<td>8.2</td>
<td>73.8</td>
</tr>
<tr>
<td>11</td>
<td>commons-codec</td>
<td>16160</td>
<td>616</td>
<td>0</td>
<td>9.7</td>
<td>11.7</td>
<td>20.7</td>
</tr>
<tr>
<td>12</td>
<td>commons-collections4</td>
<td>52040</td>
<td>13702</td>
<td>0</td>
<td>25.1</td>
<td>32.5</td>
<td>29.7</td>
</tr>
<tr>
<td>13</td>
<td>commons-discovery</td>
<td>2588</td>
<td>14</td>
<td>0</td>
<td>4.6</td>
<td>7.5</td>
<td>62.5</td>
</tr>
<tr>
<td>14</td>
<td>commons-fileupload</td>
<td>4316</td>
<td>71</td>
<td>0</td>
<td>5.7</td>
<td>9.9</td>
<td>73.8</td>
</tr>
<tr>
<td>15</td>
<td>commons-imaging</td>
<td>36657</td>
<td>93</td>
<td>0</td>
<td>30.0</td>
<td>38.9</td>
<td>29.6</td>
</tr>
<tr>
<td>16</td>
<td>commons-lang3</td>
<td>63425</td>
<td>2497</td>
<td>0</td>
<td>23.1</td>
<td>30.8</td>
<td>33.2</td>
</tr>
<tr>
<td>17</td>
<td>commons-validator</td>
<td>11982</td>
<td>416</td>
<td>0</td>
<td>7.3</td>
<td>11.0</td>
<td>51.0</td>
</tr>
<tr>
<td>18</td>
<td>compile-testing</td>
<td>1813</td>
<td>22</td>
<td>149</td>
<td>3.9</td>
<td>19.0</td>
<td>385.2</td>
</tr>
<tr>
<td>19</td>
<td>compress</td>
<td>28931</td>
<td>466</td>
<td>0</td>
<td>9.4</td>
<td>16.1</td>
<td>70.2</td>
</tr>
<tr>
<td>20</td>
<td>connector4java</td>
<td>1928</td>
<td>36</td>
<td>470</td>
<td>4.1</td>
<td>22.8</td>
<td>458.1</td>
</tr>
<tr>
<td>21</td>
<td>dbcp</td>
<td>18759</td>
<td>480</td>
<td>28</td>
<td>67.8</td>
<td>73.2</td>
<td>8.0</td>
</tr>
<tr>
<td>22</td>
<td>dropwizard-todo</td>
<td>796</td>
<td>13</td>
<td>5674</td>
<td>8.9</td>
<td>79.9</td>
<td>797.7</td>
</tr>
<tr>
<td>23</td>
<td>functor</td>
<td>21688</td>
<td>1134</td>
<td>0</td>
<td>10.7</td>
<td>18.9</td>
<td>76.4</td>
</tr>
<tr>
<td>24</td>
<td>hivemall</td>
<td>5360</td>
<td>24</td>
<td>224</td>
<td>4.7</td>
<td>9.4</td>
<td>100.1</td>
</tr>
<tr>
<td>25</td>
<td>htrace</td>
<td>1531</td>
<td>9</td>
<td>68</td>
<td>16.0</td>
<td>23.4</td>
<td>45.8</td>
</tr>
<tr>
<td>26</td>
<td>invokebinding</td>
<td>2818</td>
<td>97</td>
<td>12</td>
<td>3.9</td>
<td>6.1</td>
<td>57.2</td>
</tr>
<tr>
<td>27</td>
<td>jblas</td>
<td>12570</td>
<td>120</td>
<td>0</td>
<td>5.3</td>
<td>10.3</td>
<td>94.2</td>
</tr>
<tr>
<td>28</td>
<td>jline</td>
<td>3419</td>
<td>22</td>
<td>5707</td>
<td>3.5</td>
<td>62.9</td>
<td>1674.5</td>
</tr>
<tr>
<td>29</td>
<td>joda-time</td>
<td>82998</td>
<td>4057</td>
<td>3504</td>
<td>13.9</td>
<td>53.2</td>
<td>282.7</td>
</tr>
<tr>
<td>30</td>
<td>jpatterns</td>
<td>2604</td>
<td>33</td>
<td>4</td>
<td>3.8</td>
<td>7.4</td>
<td>94.7</td>
</tr>
<tr>
<td>31</td>
<td>jsoup</td>
<td>13556</td>
<td>413</td>
<td>48385</td>
<td>6.1</td>
<td>366.3</td>
<td>5891.9</td>
</tr>
<tr>
<td>32</td>
<td>junit</td>
<td>25916</td>
<td>867</td>
<td>0</td>
<td>12.9</td>
<td>29.1</td>
<td>125.3</td>
</tr>
<tr>
<td>33</td>
<td>laforge49</td>
<td>7245</td>
<td>56</td>
<td>185357</td>
<td>4.5</td>
<td>2133.0</td>
<td>47800.9</td>
</tr>
<tr>
<td>34</td>
<td>logback-encoder</td>
<td>637</td>
<td>18</td>
<td>1812</td>
<td>4.2</td>
<td>20.1</td>
<td>371.9</td>
</tr>
<tr>
<td>35</td>
<td>math</td>
<td>186796</td>
<td>5943</td>
<td>0</td>
<td>120.9</td>
<td>123.6</td>
<td>2.2</td>
</tr>
<tr>
<td>36</td>
<td>ogreisal,pignlpitch</td>
<td>2296</td>
<td>19</td>
<td>4200</td>
<td>64.7</td>
<td>92.1</td>
<td>42.3</td>
</tr>
<tr>
<td>37</td>
<td>paper2book</td>
<td>142</td>
<td>2</td>
<td>9329</td>
<td>2.9</td>
<td>136.7</td>
<td>4623.2</td>
</tr>
<tr>
<td>38</td>
<td>scribe-java</td>
<td>5344</td>
<td>99</td>
<td>24</td>
<td>13.0</td>
<td>16.5</td>
<td>27.2</td>
</tr>
<tr>
<td>39</td>
<td>zookeeper-utils</td>
<td>455</td>
<td>4</td>
<td>1297</td>
<td>8.5</td>
<td>18.3</td>
<td>116.1</td>
</tr>
<tr>
<td>40</td>
<td>zxing</td>
<td>42493</td>
<td>373</td>
<td>457094</td>
<td>33.8</td>
<td>4399.0</td>
<td>12904.0</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td>852021</td>
<td>36428</td>
<td>752433</td>
<td>654.8</td>
<td>9306.6</td>
<td>82941.3</td>
</tr>
<tr>
<td><strong>Arith. Mean</strong></td>
<td></td>
<td><strong>21300.5</strong></td>
<td><strong>910.7</strong></td>
<td><strong>18810.8</strong></td>
<td><strong>16.4</strong></td>
<td><strong>232.7</strong></td>
<td><strong>2073.5</strong></td>
</tr>
</tbody>
</table>

Figure 1: Time overhead of monitoring each project