Automatic Recovery from Runtime Failures

presenter name(s) removed for FERPA considerations
Key Idea

- Use automatic recovery tools to generate alternative ways to achieve the same functionality of code that can potentially fail
- Identify potential areas for failure from library calls
- Create semantically equivalent functions using different library calls
- Employ tool to avoid failures at run-time
Study Questions

- Is automatic recovery effective in making applications more resilient to faults?
- Are the techniques efficient enough to be practically usable?
- Is modular software to some significant extent intrinsically redundant?
Why?

● Their argument: Modern software is intrinsically redundant
● Redundancy is an intrinsic property of modular software
● Implementations tend to be semantically equivalent
● Use of libraries
  ○ Backwards compatible functionality equivalence
  ○ Cross-library semantic equivalence
  ○ Functional equivalence for different use cases (based on input)
● The point is to find operations equivalent in intended behavior, but not in actual observable behavior
Equivalence of Tested Libraries

**TABLE I**
**EQUIVALENT SEQUENCES FOUND IN REPRESENTATIVE JAVA LIBRARIES**

<table>
<thead>
<tr>
<th>Library</th>
<th>Guava</th>
<th>SWT</th>
<th>JodaTime</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classes considered</td>
<td>116</td>
<td>252</td>
<td>12</td>
</tr>
<tr>
<td>Total equivalences found</td>
<td>1715</td>
<td>1494</td>
<td>135</td>
</tr>
<tr>
<td>Average per class</td>
<td>14.78</td>
<td>5.93</td>
<td>11.25</td>
</tr>
</tbody>
</table>
How it Works - ARMOR

● Preprocessor identifies Roll-Back Areas (RBAs)
  ○ Static Analysis
  ○ Calls to the library that could be re-written
  ○ May result in Failure but minimal
  ○ Supports nested RBAs
  ○ Creates checkpoints before RBAs
  ○ Method bodies / singular field initialization expression (encapsulated as a method)

● Rewrites RBAs and compiles them
● Each new solution wrapped in a loop based on passing checkpoints
● On failure loop iterates to next available solution for implementation
  ○ Based on past success
● Rolls back to last checkpoint if notified of failure
● Runs until failure-free or runs out of solutions
Example: JodaTime

```
  // failing operation
  DateTime beginDay = dt.millisOfDay().withMinimumValue();

  // workaround 1
  DateTime beginDay = dt.toDateMidnight().toDateTime();

  // workaround 2
  DateTime beginDay = dt.withTimeAtStartOfDay();
```

Listing 2. Workarounds for issue n. 3304757 of JodaTime
```java
class CurrentMidnight {
    DateTimeZone tz = DateTimeZone.forID("America/Sao_Paulo");
    DateTime midnight;

    public void initDayAndZone()
    {
        DateTimeZone.setDefault(tz);
        DateTime dt = new DateTime();
        ...
        setMidnight(dt);
    }

    private void setMidnight(DateTime dt)
    {
        midnight = dt.millisOfDay().withMinimumValue();
    }

    public DateTime getMidnight()
    {
        return midnight;
    }
}

class Main {
    public static void main(String args[])
    {
        ...
        CurrentMidnight cm = new CurrentMidnight();
        cm.initDayAndZone();
        ...
    }
}
```

Listing 3. Example application code
class CurrentMidnight {
    DateTimeZone tz = tz_init();
    public DateTimeZone tz_init_original() {
        return DateTimeZone.id("America/Sao_Paulo");
    }
    public DateTimeZone tz_init() {
        try {
            create_checkpoint();
            return tz_init_original();
        } catch (Exception ex) {
            while (more_rba_variants_available) {
                try {
                    restore_checkpoint();
                    load_new_rba_variant();
                    return tz_init_original();
                } catch (Exception ex1) {
                    // record variant failure and adjust priorities
                    ...
                }
            }
            throw ex;
        } finally {
            discard_checkpoint();
        }
    }
    public DateTime midnight;
    // initDayAndZone proxy method not shown
    public void setMidnight_original(DateTime dt) {
        midnight = dt.millisOfDay();
    }
    public void setMidnight(DateTime dt) {
        try {
            create_checkpoint();
            setMidnight_original(dt);
        } catch (Exception ex) {
            boolean success = false;
            while (!success && more_rba_variants_available) {
                try {
                    restore_checkpoint();
                    load_new_rba_variant();
                    setMidnight_original(dt);
                    success = true;
                } catch (Exception ex1) {
                    // record variant failure and adjust priorities
                    ...
                }
            }
            if (!success) throw ex;
        } finally {
            discard_checkpoint();
        }
    }
}
The Experiment

- **Libraries used**
  - JodaTime: Library of utility functions to represent and manipulate dates and time.
  - Guava: The Google “core” library for collections, I/O, caching, concurrency, string processing, etc.

- **Application Software**
  - Fb2pdf: A command-line utility to convert files from the FB2 e-book format into PDF. Fb2pdf uses the Java date/time library but they changed it to use the fully compatible JodaTime library.
  - Carrot2: A search results clustering engine. Carrot2 uses the Guava library.
  - Caliper: A framework for writing, running and viewing the results of Java microbenchmarks. Caliper uses the Guava library.
Experiment cont.

- Formulated code re-writing rules and ran ARMOR preprocessor

### TABLE II

**RESULTS OF THE PREPROCESSING ON THE SELECTED APPLICATIONS**

<table>
<thead>
<tr>
<th>Application</th>
<th>Caliper</th>
<th>Carrot2</th>
<th>Closure</th>
<th>Fb2pdf</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total RBAs</td>
<td>130</td>
<td>139</td>
<td>2099</td>
<td>17</td>
</tr>
<tr>
<td>RBAs with variants</td>
<td>60</td>
<td>106</td>
<td>687</td>
<td>17</td>
</tr>
</tbody>
</table>
Experiment cont.

- Conducted a more extensive evaluation using seeded faults with both libraries and all four applications
- Ran mutation generation on all programs
- Used Daikon to get invariants
  - Used invariants that worked in the original code but failed in RBAs in mutated code
  - Added these as assertions in RBA
### Results

#### TABLE III
**Mutation analysis and effectiveness of ARMOR**

<table>
<thead>
<tr>
<th>Mutants</th>
<th>Caliper</th>
<th>Carrot2</th>
<th>Closure</th>
<th>Fb2pdf</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total mutants</td>
<td>21297</td>
<td>21297</td>
<td>21297</td>
<td>16858</td>
</tr>
<tr>
<td>Relevant mutants</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Equivalent</td>
<td>309</td>
<td>187</td>
<td>344</td>
<td>2200</td>
</tr>
<tr>
<td>Non-equivalent detected</td>
<td>210</td>
<td>120</td>
<td>177</td>
<td>1805</td>
</tr>
<tr>
<td>Success</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Success not detected</td>
<td>0</td>
<td>8</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Loop</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Detected</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Loop not detected</td>
<td>12</td>
<td>9</td>
<td>15</td>
<td>47</td>
</tr>
<tr>
<td>Total mutants run with ARMOR</td>
<td>87</td>
<td>47</td>
<td>149</td>
<td>347</td>
</tr>
<tr>
<td>Mutants where ARMOR is successful</td>
<td>(28%) 24</td>
<td>(48%) 24</td>
<td>(47%) 70</td>
<td>(19%) 67</td>
</tr>
</tbody>
</table>

#### TABLE IV
**Overhead incurred by ARMOR in normal non-failing executions (median over 10 runs)**

<table>
<thead>
<tr>
<th>Time (seconds)</th>
<th>Caliper</th>
<th>Carrot2</th>
<th>Closure</th>
<th>Fb2pdf</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original total running time</td>
<td>30.13</td>
<td>2.43</td>
<td>5.40</td>
<td>2.26</td>
</tr>
<tr>
<td>Exception-handling only (no checkpoints)</td>
<td>(1%) 30.41</td>
<td>(69%) 4.15</td>
<td>(95%) 10.53</td>
<td>(68%) 3.79</td>
</tr>
<tr>
<td>Snapshot-based checkpoints</td>
<td>31.78</td>
<td>(117%) 5.32</td>
<td>&gt;1h</td>
<td>(121%) 4.99</td>
</tr>
<tr>
<td>Change-log-based checkpoints</td>
<td>(2%) 30.87</td>
<td>(94%) 4.75</td>
<td>(194%) 15.90</td>
<td>(114%) 4.70</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Memory (MB)</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Original total memory allocated</td>
<td>1.40</td>
<td>8.87</td>
<td>30.56</td>
<td>17.90</td>
</tr>
<tr>
<td>Snapshot-based checkpoints</td>
<td>12.30</td>
<td>23.78</td>
<td>—</td>
<td>90.94</td>
</tr>
<tr>
<td>Change-log-based checkpoints</td>
<td>10.18</td>
<td>11.37</td>
<td>120.58</td>
<td>25.93</td>
</tr>
</tbody>
</table>

| Number of recorded checkpoints (approx.) | 30 | 2,350 | 1,255,000 | 4 |
| Values saved in change-log-based checkpoints (approx.) | 26,000 | 270,000 | 1,880,000 | 9,000 |
Fb2pdf had the smallest amount of RBAs, largest amount of mutants affecting execution, and lowest success rate
  - Applications with calls to the library that make up a larger portion of the library code
Carrot2 had failed for one fault
  - Library call within a library code
  - High overhead with regard to checkpoint stack
Overall, author asserts that the program results are positive
In the future, to solve more issues deep in library code / test on different libraries
What’s New?

- Others have exploited redundancy
  - Requires additional code
  - Cabral - requires written exception handlers
  - Chang - manually write patches
  - Harmanci - code alternative code blocks
- Is different, as it exploits redundancy already available in libraries
  - Less design cost
- General purpose
- Designed to work on deployed applications
  - Prevent mistakes at run-time
- The notion of intrinsic redundancy in modern software applications
Discussion

1. Could this be potentially dangerous if implemented for a live system built on several nested library dependency?
2. Is the overhead worth it?
3. As library redundancy increases, will this process become more and more expensive?
4. Worth it for smaller / newer libraries / lower-level code?
5. Can we use this process to replace non-library code with semantically equivalent syntax?