On the Use of Specification-Based Assertions as Test Oracles

David Coppit
The College of William and Mary

Jennifer Haddox-Schatz
Daniel H. Wagner Associates, Inc.
The Oracle Problem

- Testing widely used for software verification
  - Test case: input and expected output

- The *oracle problem*: how can we compute the expected output?

- Oracle implementations costly, untrustworthy

- Typical solution is manual development
  - Costly, error-prone → Limits amount of testing
  - Not a general solution
Potential Solution: Assertions

• *Assertion*: a self-check on the running state of a program

• Usually easier to check a result than compute one
  • Factor deterministically in $O(c (\log n)^{1/3} (\log \log n)^{2/3})$ time, check in $O(\log n)$ time

• But assertions usually haphazardly used to check implementation details ($\text{assert}(p \neq \text{null})$)
Research Question

Can “strong” assertions effectively reveal faults during testing, sidestepping the oracle problem?

- Answer is checked instead of computed
  - Often simpler than separate oracle implementation
- Checker is independent of implementation
  - Different algorithm $\rightarrow$ less chance for common mode faults (N-version programming problem)
  - Works for any implementation
Evaluating Effectiveness

1. Develop and validate a formal specification
2. Convert the specification into assertions for a specification-derived assertion version (SDAV)
3. Inject faults or use pre-existing faults
4. For each fault:
   1. Execute the existing implementation (with programmer assertions) with test cases
   2. Execute the SDAV with the same test cases
   3. Compare the fault-revealing effectiveness of both versions
Example: SDAV of Sort

1. Formal specification in Z [Spivey]

\[ \forall \exists Sort : \text{seq } \mathbb{Z} \rightarrow \text{seq } \mathbb{Z} \]
\[ \langle \_ \rangle \]
\[ \forall \exists \text{in, out : seq } \mathbb{Z} \cdot \]
\[ \forall \exists \text{Sort(in)} = \text{out} \iff \]
\[ \forall \exists \text{items(in)} = \text{items(out)} \quad \text{¶} \]
\[ \forall \exists (\forall i,j : \text{dom out} \mid i < j \cdot \text{out}(i) \neq \text{out}(j))) \]
Example: SDAV of Sort (cont)

2. Implementation in C, with assertions

```c
void Sort(int numbers[], int array_size) {
    // Save a copy of the list for post-condition assertion
    int old_numbers[] = copy_array(numbers[], array_size);
    
    for (int i = 0; i < array_size; i++) {
        int save = numbers[i];
        for (int j = i ; j > 0 && (numbers[j-1] > save) ; j--)
            numbers[j] = numbers[j-1];
        numbers[j] = save;
    }
    // New assertions
    assert( items(old_numbers, array_size) == items(numbers, array_size) );
    for (int i = 0; i < array_size-1; i++)
        for (int j = i+1; j < array_size-1; j++)
            assert( numbers[i] <= numbers[j] );
}
```
Two Case Studies

- **Auger**: interface for submission of batch jobs to a server farm at Jefferson Laboratory
  - Job management package: provides intermediate representation of a job submission from a user script
  - 21,529 test inputs; some programmer assertions

- **TDoG**: test driver generator tool from Cigital
  - Data Recorder: decomposes arbitrary Java objects into textual form for inspection by the user
  - We developed 27 test cases; some programmer assertions
Developing the Assertions

1. Created Object-Z specifications for case studies after careful study of documentation
   • Validated specification informally with developers and with type checker

2. Derived assertions from the specifications and embedded them using *Jass* assertion tool
   • SDAV has new assertions, but no programmer asserts
   • Original version has test cases, programmer asserts
Testing

- Both systems were stable, and we didn’t have access to historical faults

3. Use *fault injection*: Perturb the inputs or output of methods
   - Injected faults automatically with JavaWrap
   - Injected faults manually as well

4. Compared effectiveness
   - Original version (OV), SDAV, or both OV+SDAV
Quantitative Results: Auger

<table>
<thead>
<tr>
<th>Fault</th>
<th>Fault Injection</th>
<th>OV Only</th>
<th>OV + SDAV</th>
<th>SDAV Only</th>
</tr>
</thead>
<tbody>
<tr>
<td>Increment Job ID by 10</td>
<td>21,529</td>
<td>0</td>
<td>0</td>
<td>21,529</td>
</tr>
<tr>
<td>Decrement WorkJob Height by 10</td>
<td>21,529</td>
<td>0</td>
<td>0</td>
<td>21,529</td>
</tr>
<tr>
<td>Decrement WorkJob ID by 10</td>
<td>21,529</td>
<td>0</td>
<td>0</td>
<td>21,529</td>
</tr>
<tr>
<td>Remove from Job List</td>
<td>21,529</td>
<td>0</td>
<td>0</td>
<td>21,529</td>
</tr>
<tr>
<td>Replace Element in Job List</td>
<td>21,529</td>
<td>0</td>
<td>0</td>
<td>21,529</td>
</tr>
<tr>
<td>Add Element to Dependency List</td>
<td>21,529</td>
<td>0</td>
<td>0</td>
<td>21,529</td>
</tr>
<tr>
<td>Decrement JobSubmission ID by 10</td>
<td>17,403</td>
<td>0</td>
<td>0</td>
<td>17,403</td>
</tr>
<tr>
<td>Decrement FileJob Height by 10</td>
<td>10,479</td>
<td>0</td>
<td>0</td>
<td>10,479</td>
</tr>
<tr>
<td>Decrement FileJob ID by 10</td>
<td>10,479</td>
<td>0</td>
<td>0</td>
<td>10,479</td>
</tr>
<tr>
<td>Remove from Dependency List</td>
<td>10,479</td>
<td>0</td>
<td>0</td>
<td>10,479</td>
</tr>
<tr>
<td>Add to Empty Dependency List</td>
<td>10,423</td>
<td>0</td>
<td>0</td>
<td>10,423</td>
</tr>
<tr>
<td>Use Substring of Command</td>
<td>21,529</td>
<td>0</td>
<td>0</td>
<td>5</td>
</tr>
</tbody>
</table>
### Quantitative Results: TDoG

<table>
<thead>
<tr>
<th>Fault</th>
<th>Fault Injection</th>
<th>OV Only</th>
<th>OV + SDAV</th>
<th>SDAV Only</th>
</tr>
</thead>
<tbody>
<tr>
<td>Negate Recursion Depth</td>
<td>10</td>
<td>0</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>Negate Array Dimension</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>Empty Field Name</td>
<td>16</td>
<td>0</td>
<td>0</td>
<td>16</td>
</tr>
<tr>
<td>Fix Negative Hash Value</td>
<td>22</td>
<td>0</td>
<td>0</td>
<td>22</td>
</tr>
<tr>
<td>Empty Object Type Name</td>
<td>25</td>
<td>0</td>
<td>0</td>
<td>25</td>
</tr>
<tr>
<td>Corrupt Unique ID</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>Null Primitive Value</td>
<td>22</td>
<td>0</td>
<td>22</td>
<td>0</td>
</tr>
<tr>
<td>Omit Null Check</td>
<td>10</td>
<td>0</td>
<td>10</td>
<td>0</td>
</tr>
</tbody>
</table>
Qualitative Results: Assertion Difficulties

- Assertion language somewhat weak
  - Jass not much better than `assert`
  - Need full *design by contract* [Meyer] support

- Specification $\rightarrow$ implementation gap
  - Size constraints on data types, potential null values
  - Specification assumes objects always obey invariant, but OO implementations often allow incoherence
public class JobSubmission {
    private List jobs;
    private String script;
    private boolean isCoherent;
    public JobSubmission() {
        isCoherent = false;
        ...
    }
    ...
    public void objectCoherent() {
        isCoherent = true;
    }
    /** invariant forall i: { 0 .. jobs.size()-1 } #
        ( (Job)jobs.get(i) instanceof WorkJob
        ||(Job)jobs.get(i) instanceof FileJob ) 
        &&
        ( !isCoherent || script != null ) **/
Evaluation of Spec-Derived Assertions

- **Benefits:**
  - Can reveal invalid program states when normal assertions and tests do not
  - Remove the need to manually derive expected outputs, or implement an oracle
  - Don’t need to build test scaffolding to extract outputs

- **Downsides:**
  - Developing a specification is costly
  - Verifying assertions can be nontrivial
  - Suffers from specification abstraction, language gaps
Evaluation of Experiments

• Case study code not very large
  • Auger: 834 LOC, TDoG: 3,475 LOC

• Case studies were very mature
  • Auger already run with 21,529 test input scripts
  • TDoG data recorder: code reviews, tested well
  • Had to use fault injection

• Specifications had to be created “after the fact”
Current and Future Work

- Larger systems, no fault injection
  - Nova solver: 12K SLOC, 14 real faults, full C/D coverage of the test suite

- Characterize cost tradeoff
  - Direct implementation of assertions without spec.
  - Partial assertions
  - Infer assertions from runtime behavior
Conclusion

• Evaluated the fault detection capability of specification-derived assertions
  • Two case studies on real, but small systems

• Results are positive:
  • Higher quality assertions appear to be better
  • Can perhaps replace oracles

• Experiences highlight the difficulties of writing effective assertions, and provide some strategies for coping with them
Thank You