

## On the Use of Specification-Based Assertions as Test Oracles

---

David Coppit

Jennifer Haddox-Schatz

The College of  
William and Mary

Daniel H. Wagner  
Associates, Inc.



4/8/05

## 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

4/8/05

2

## 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  $O(e^{c(\log n)^{1/3}(\log \log n)^{2/3}})$  time, check in  $O(\log n)$  time
- But assertions usually haphazardly used to check implementation details (`assert(p != null)`)

4/8/05

3

## 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 → less chance for common mode faults (N-version programming problem)
  - Works for any implementation

4/8/05

4

## 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

4/8/05

5

## Example: SDAV of Sort

1. Formal specification in Z [Spivey]

$$\text{Sort} : \text{seq } \mathbb{Z} \rightarrow \text{seq } \mathbb{Z}$$
$$\forall in, out : \text{seq } \mathbb{Z} .$$
$$\text{Sort}(in) = out \Leftrightarrow$$
$$\text{items}(in) = \text{items}(out) \wedge$$
$$(\forall i, j : \text{dom } out \mid i < j \cdot out(i) \leq out(j))$$

4/8/05

6

## Example: SDAV of Sort (cont)

2. Implementation in C, with assertions

```
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] );
}
```

4/8/05

7

## 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

4/8/05

8

## Developing the Assertions

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

4/8/05

9

## Testing

- Both systems were stable, and we didn't have access to historical faults
- Use *fault injection*: Perturb the inputs or output of methods
    - Injected faults automatically with JavaWrap
    - Injected faults manually as well
  - Compared effectiveness
    - Original version (OV), SDAV, or both OV+SDAV

4/8/05

10

## Quantitative Results: Auger

Fault	Fault Injection	OV Only	OV + SDAV	SDAV Only
Increment Job ID by 10	21,529	0	0	21,529
Decrement WorkJob Height by 10	21,529	0	0	21,529
Decrement WorkJob ID by 10	21,529	0	0	21,529
Remove from Job List	21,529	0	0	21,529
Replace Element in Job List	21,529	0	0	21,529
Add Element to Dependency List	21,529	0	0	21,529
Decrement JobSubmission ID by 10	17,403	0	0	17,403
Decrement FileJob Height by 10	10,479	0	0	10,479
Decrement FileJob ID by 10	10,479	0	0	10,479
Remove from Dependency List	10,479	0	0	10,479
Add to Empty Dependency List	10,423	0	0	10,423
Use Substring of Command	21,529	0	0	5

4/8/05

11

## Quantitative Results: TDoG

Fault	Fault Injection	OV Only	OV + SDAV	SDAV Only
Negate Recursion Depth	10	0	10	0
Negate Array Dimension	10	0	0	10
Empty Field Name	16	0	0	16
Fix Negative Hash Value	22	0	0	22
Empty Object Type Name	25	0	0	25
Corrupt Unique ID	3	0	0	3
Null Primitive Value	22	0	22	0
Omit Null Check	10	0	10	0

4/8/05

12

## Qualitative Results: Assertion Difficulties

---

- Assertion language somewhat weak
  - Jass not much better than `assert`
  - Need full *design by contract* [Meyer] support
- Specification → implementation gap
  - Size constraints on data types, potential null values
  - Specification assumes objects always obey invariant, but OO implementations often allow incoherence

4/8/05

13

## Solution: Object Coherence Pattern

---

```
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 ) */
}
```

4/8/05

14

## 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

4/8/05

15

## 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"

4/8/05

16

## 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

4/8/05

17

## 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

4/8/05

18

***Thank You***