

# An Evaluation of Specification-Derived Assertions

Matthew F. Curtis-Maury and  
Jennifer M. Haddox-Schatz

1

## Outline

- Introduction: Problem Statement & Approach
- Related Work
- Using Formal Specifications to Derive Assertions
- Experiment: Nova Solver
- Results
- Evaluation
- Conclusion

2

## Introduction

3

## Problem Statement

- Assertions aid the process of software testing
  - Provide self-check on program's data state
  - No longer solely dependent on output
- Assertions not always used effectively
  - Used to check low-level details
  - Not considered until implementation stage

4

## Approach

- Derive assertions directly from a program's formal specification
  - Formal specification provides precise description of program's intended behavior
  - Provides logical foundation from which to develop assertions
- Goal: create assertions capable of verifying correctness of overall system specification

5

## Related Work

- Voas [1994, 1995, 1997, 1999]:
  - Place assertions in code locations least likely to reveal faults
- Rosenblum [1992]:
  - APP; developer experiences with assertions
  - Suggests formal specifications for deriving assertions
- Meyer [1992]:
  - Design By Contract
  - Shift assertion creation to the design phase

6

## Using Formal Specifications to Derive Assertions

7

## Example: Sort

- ```
Public void sort(Element a[], int size) {  
    assert(a!=null)  
    // Implementation ...  
}
```
- Assume the implementation sorts the list in increasing order
- Assertion alone can not guarantee correctness of the implementation
- Need stronger assertions

8

## Specification for Sort

- Consider a formal specification for the sort implementation:

$Sort : seqElement \rightarrow seqElement$

$\forall in, out : seqElement \bullet$

$Sort(in) = out \Leftrightarrow$

$items(in) = items(out) \wedge$

$(\forall i, j : 1..#out \mid i < j \bullet out(i) \leq out(j))$

## Deriving Assertions

$items(in) = items(out)$

- Derive the following assertion:  

```
assert(isPermutation(old_a, a, size))  
  
( $\forall i, j : 1..#out \mid i < j \bullet out(i) \leq out(j)$ )
```
- Derive the following assertion:  

```
for(int i=0; i<size-1; i++) {  
    assert(a[i]<=a[i+1])  
}
```

10

## Integrate Assertions and Original Code

- Add assertions to end of sort method

```
Public void sort(Element a[], int size) {  
    Element old_a=a; // make copy of list  
    assert(a!=null)  
    // Implementation ...  
  
    assert(isPermutation(old_a, a, size));  
    for(int i=0; i<size-1; i++) {  
        assert(a[i]<=a[i+1])  
    }  
}
```

- New assertions guarantee correct output, and are implementation-independent

11

## Experiment: Nova Solver

12

## Experiment

- Case study: Nova Solver and its specification
  - a Fault Tree analysis tool.
  - Program to predict the unreliability of a system.
- Applying approach required understanding of
  - Code
  - Z specification
  - existing test suite

13

## Experiment

- Add formal-specification derived assertions.
- Add programmer-created assertions.
- Run test suite and record assertion failures.
- Compare results from each type of assertion.

14

## Testing

- Small test suite existed.
- We created additional tests of two types:
  - System tests.
  - Unit tests.
- Our tests were largely composed of illegal fault trees and invalid objects.

15

## Results

16

## Costs of Implementation

- Programmer-created assertions.
  - 6 weeks to implement; one person
  - 150 lines of code
  - Simple assertions
- Formal-specification derived assertions.
  - 8 weeks; 3 people
  - 2300 LOC
  - Complex assertions, had to be debugged.
- Greater time and effort requirements for a project.

17

## Existing Test Suite Results

- No assertion failures in programmer assertions.
- Two errors found in specification-derived assertions.
  - Threshold and Time classes
- Errors represent code deviations from the specification.

18

## Additional Test Suite Results

- One assertion failure in programmer assertions.
  - Fault Tree copy constructor.
- 10 system test & 20 unit test failures in specification-derived assertions.
  - Illegal fault trees.
  - Invalid objects.
- Illegal input caused failures, but results are still interesting.

19

## Evaluation

20

## Cost-Effectiveness

- Large undertaking.
- High cost may make it inappropriate for non-critical systems.
- May still be worthwhile for life-critical systems.
  - Pacemaker software.
  - Flight control system.

21

## Limitations

- May not catch low-level errors.
- Requires that a formal specification be written for the system.
- System must also follow the structure of the specification.

22

## Benefits

- Help find code inconsistencies with the specification.
- Specification-derived assertions caught illegal objects.
- Provides error-checking within each unit.

23

## Evaluation of Experiment

- Lack of errors found is not surprising.
  - Stable software system.
  - Small test suite.
  - Relatively small case study program.
- Experiment would optimally be performed during the development of a real system.
- Nova Solver was more realistic than developing a toy system.

24

## Conclusions

25

## Future Work

- Tested with a larger test suite.
  - Two group members are applying TestEra.
- More experience reports on deriving assertions from formal specifications.
- Run similar test on a real system during the development process.
- Perhaps this process could be automated.

26

## Summary

- Effective software testing requires more research.
- We have proposed a new approach for the creation of assertions.
- We have highlighted the benefits and limitations of this approach.

27