

CSci 780
Advanced Software Engineering

**Software Assurance by
Bounded Exhaustive Testing**

by

**Kevin Sullivan, Jinlin Yang, David Coppit,
Sarfraz Khurshid, and Daniel Jackson**

9/20/04

1

Context

- Two key difficulties in software engineering:
 - Are we building the right thing? (validation)
 - Are we building the thing right? (verification)
- Need the specification to answer:
 - What to test?
 - What should the result be?
- How to use specification to verify software?

9/20/04

2

Current State of the Practice

- Full formal specification is too costly
- Instead people test known problem inputs
 - Test cases are a partial specification of "correct"
- Problems with traditional testing:
 - Human-intensive
 - Small number of test cases
 - Doesn't address subtle "unexpected" faults

9/20/04

3

Example: Logical Expressions

- Consider the inputs to a program to evaluate logical expressions:
 - Each operator: $T \vee F$, $T \wedge F$, $T \Rightarrow F$
 - Grouping: $(T \Rightarrow F) \vee T$
 - Unary operator (negated values): $\neg T \vee F$
- What else?
 - Operator precedence: $3 \Rightarrow 4 \Rightarrow 2$
 - More???

9/20/04

4

Problem

- *Verifying the correctness of software is difficult, especially with respect to subtle faults, which elude traditional informal testing*
- Examples:
 - Therac-25 overdoses
 - Denver airport radar outage
 - Mars Pathfinder priority inversion

9/20/04

5

Outline

- *Motivation*
- Revisiting a Stupid Idea: Exhaustive Testing
- Approach
 - Oracle
 - Automatic Generation of Structured Inputs
- Key Challenges
- Evaluation
- Conclusion
- Discussion

9/20/04

6

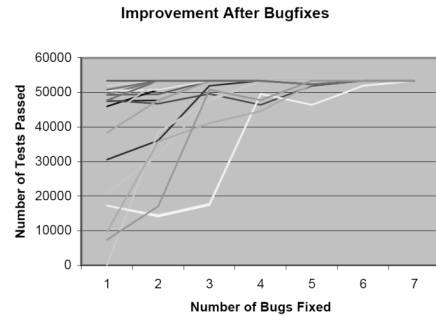
A Stupid Idea: Exhaustive Testing

- Exhaustive testing: Test all inputs
- Long ago dismissed as infeasible
- But:
 - Moore's Law
 - Works okay for model checking
 - No need to identify "good" inputs to test
 - Many more test cases
 - If automated, no human effort
 - Can reveal subtle faults

9/20/04

7

But Is It Really Stupid?



9/20/04

8

Requirements for Exhaustive Testing

- Enumeration strategy
- An input generator
- An oracle to check the results
- Automated testing infrastructure
 1. Generate input
 2. Execute program
 3. Check result

9/20/04

9

Approach

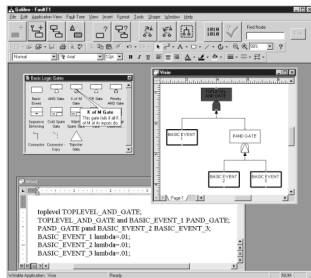
1. Selectively reverse engineer the program
 - Specification of the input
 - Partial specification of behavior
2. Implement behavior spec. as trustworthy oracle
3. Generate inputs using TestEra+input spec.
4. Run program and oracle, and compare results
5. Resolve differences
 - Fix specification, oracle, or program
 - Don't fix. Mask common-cause failures

9/20/04

10

Case Study: Galileo Reliability tool

- Developed at Uva with NASA
- Built out of mass-market applications
- Now used by NASA ISS (Need dependability)

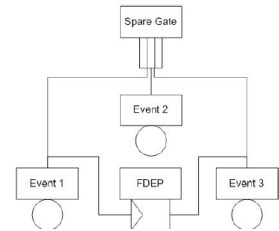


9/20/04

11

Dynamic Fault Trees

- Reliability modeling notation
 - Events + Constraints
- Solve to compute unreliability, sensitivity
 - static solver
 - dynamic solver



9/20/04

12

1: Reverse Engineering the Specification

- Ignored GUI, focused on subtle DFT semantics
- Specified in Z [Spivey 1992]
- Four months initial development (part time)
- Three months more validating (part time)
- Revised the language (not exactly Galileo's)
 - Fixed semantic problems
 - Improved regularity, orthogonality

9/20/04

13

2: Building The Oracle

- Nova
- Implemented essential semantics in 3 months
- Directly from specification
- Goal: easy verification
 - No optimization, no OO, etc.
- Runs slowly for large inputs (Galileo 5x faster)
- 10,680 SLOC

9/20/04

14

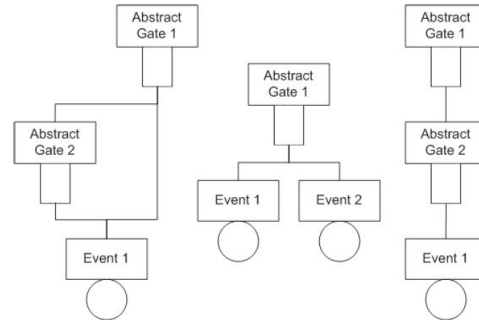
3: Generation of Inputs Using TestEra

- Two-step process
 - Used Alloy to generate *abstract* FTs (structure)
 - Wrote a Perl program to instantiate nodes for *concrete* FTs to test
- Translated Z input specification into Alloy
- Implemented an Alloy->Text DFT model translator in Java
- Symmetry breaking
- Wrote Perl postprocessor to prepare text DFT for Galileo and Nova

9/20/04

15

Example: AFTs (3 Nodes, No Constraints)



9/20/04

16

Input Explosion

Events	Seqs	FDeps	AFTs	DFTs
3	0	0	3	48
3	0	1	4	56
3	1	0	6	96
3	1	1	8	112
4	0	0	16	1,571
4	0	1	46	3,614
4	1	0	192	18,852
4	1	1	552	43,368
5	0	0	176	186,668
5	0	1	717	616,806
5	1	0	10,560	11,201,520
5	1	1	43,020	37,017,000
6	0	0	4,229	93,454,072
7	0	0	230,470	astronomical

9/20/04

17

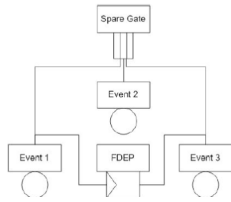
4: Execution of Inputs

- 75 hours to generate all FTs with 7 nodes
 - 200,000 concrete FTs
 - 1GHz, 1GM Linux
- 1 week to test first 250,000 inputs (4 nodes)
 - Iterative process: mask faults or fix them
 - Would take 355 years to run 93M FTs of 6 nodes
 - Dual-CPU 3GHz, 1GB Windows

9/20/04

18

5: Resolving Differences 1/3

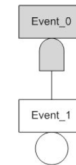


- 8 Galileo faults
 - Example above: after Event 1 failed, Spare Gate was failed even though Event 2 was still operational

9/20/04

19

5: Resolving Differences 2/3



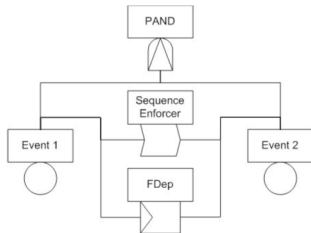
*toplevel Event_0;
Event_0 and Event_1;
Event_1 lambda=.01 cov=0 res=.5 repl= 2 dorm=.5;*

- 3 specification errors
 - Example: Hidden assumption about failure rates

9/20/04

20

5: Resolving Differences 3/3



- 3 Nova faults
 - Sequence enforcer not implemented

9/20/04

21

Technical Challenges

- Limitations of TestEra
 - Had to use abstraction + concretization
- Input explosion
 - Symmetry breaking to remove isomorphic DFTs
 - Bound the values of inputs (e.g. failure rates)
- Oracle speed
 - Smallest inputs first
- Systematic failures
 - Mark DFTs that are expected to fail

9/20/04

22

Evaluation

- Cost-effective use of formal specifications
- Sheer number of test cases reveals subtle faults
- Automated testing
- Coincident faults
- Partial oracle may not be sufficiently strong
- Need to evaluate BET as a coverage criterion
- Need to tie results to software reliability

9/20/04

23

Selected Related Work

- TestEra [2001]: Translate inputs from Alloy, run code, translate result to Alloy and check
- Korat [2002]: Use method specs to generate inputs and check results
- Small scope hypothesis [Marinov et. al 2003]: exhaustive outperforms random testing
- Traditional coverage criteria [Weyuker 1985]: based on statement, branch, condition coverage

9/20/04

24

Conclusion

- Approach shows promise
- Were able to overcome technical difficulties and find real faults
 - Specification and implementation
 - Implementation and oracle cross-check each other
- Structurally complex inputs, sizeable code
- Time to reconsider bounded exhaustive testing

9/20/04

25

Discussion

- Which is more important: exhaustiveness or size? Alternative input generation strategies?
- Problems with approach?
- Potential solutions to these problems?
- Future research?

9/20/04

26