SoHaR

# Testing for Software Safety

Presented by
Herbert Hecht
SoHaR Incorporated
Beverly Hills, California

NASA Ames Research Center
September, 2000

Dale Carnegie® Training®

SoHaR

# Software Testing is Different

- Meaningless: life/wear, environmental extremes, variability
- Redundancy must use dissimilar elements
- Exception handling ⇔ Overload testing (with limitations)

# Two Aspects of Software Safety

- Functional software should not generate hazards

*Guidance should not steer into the ocean*

- Monitoring programs must perform flawlessly

*Start back-up computer when primary fails*

# Critical Issues in Monitoring Programs

SoHaR

- Invoked under unusual conditions
- Impossible to generate a complete scenario
- Defense-in-depth may have been breached
- Running in shared environment

# NASA MUX (1990/91)

Multi-University Experiment

- Determine probability of correlated errors in multi-version programs
- Generate acceleration vector from 6 non-orthogonal accelerometers
- 20 teams from Univ. of California, Illinois, Virginia and NC State

# Results from Independent Test of MUX Programs

| No. of Anomalies | Failure Fraction |
|---|---|
| 0 | 0.001 |
| 1 | 0.01 |
| 2 | 0.13 |
| 3 | 0.58 |

Eckhardt et al., "Strategy for Improving Reliability" *IEEE Transactions of Software Engineering* July 1991, p. 692-702

# Similar Findings

- **Post-Challenger Shuttle Software**
- **Deep Space Network Telemetry Software**
- **Telephone Switching Software**
- **Nuclear Reactor Protection Software**

H. Hecht and P. Crane, "Rare Conditions and their Effect on Software Failures", *Proc. 1994 RAMS*, pp. 334 - 337

# Evaluation of Results

- Consequences of multiple failures are difficult to analyze
- Testers not motivated to construct test cases for multiple failures
- Multiple failures are a major software safety hazard

**Emphasize testing for multiple failures**

# More Intelligent Testing

***Strategy***

- Testing can not be “complete”
- Budgets are limited
- Scenarios rich in multiple failures will reduce cost per defect found

# More Intelligent Testing

***Test case selection (Subdomains)***

- Use local failure data
- Use local test effectiveness data
- Use global data where available
- Document selection criteria

# Test Effectiveness for Safety

- User profile testing not applicable to safety
- Undetected hazard vs. false inhibit
- Random vs. selected test cases
- Safety functions in shared computer

***A neglected topic - Research needed***

# Where we are now - Procedures

- Project oriented safety reviews
- Few specific guidelines for software test
- Test reports show that system is safe but little of test conduct and effectiveness

# Where we are now - Activities

- Test case selection is empirical, should emphasize multiple exceptions
- No recognized figure-of-merit for test effectiveness
- No organized data exchange

***Large resources required for test of software safety***

# Where we want to be

- Documented test conduct and effectiveness
- Data exchange for test effectiveness evaluation
- Best practices for software safety testing

***Better test rationale - fewer resources***

# How to get there

- Scope current expenditures for testing of software safety
- Establish research budget at x% of annual software test expenditure
- Research goals
  - Define test figure-of-merit
  - Establish data exchange format
  - Analyze data and generate best practices

SoHaR

# Conclusions

- Many open issues in testing for software safety
- Testing for software safety is largely empirical
- Little data exchange
- Software test research is seldom applicable to monitoring programs

SoHaR

# Recommendations

- Fund research
- Require reporting of test case selection criteria and their effectiveness
- Encourage dissemination of test effort and results data