
# Combining Risk Analysis and Slicing for Test Reduction in Open Architecture

Valdis Berzins

Naval Postgraduate School

# U.S. Navy Open Systems Architecture

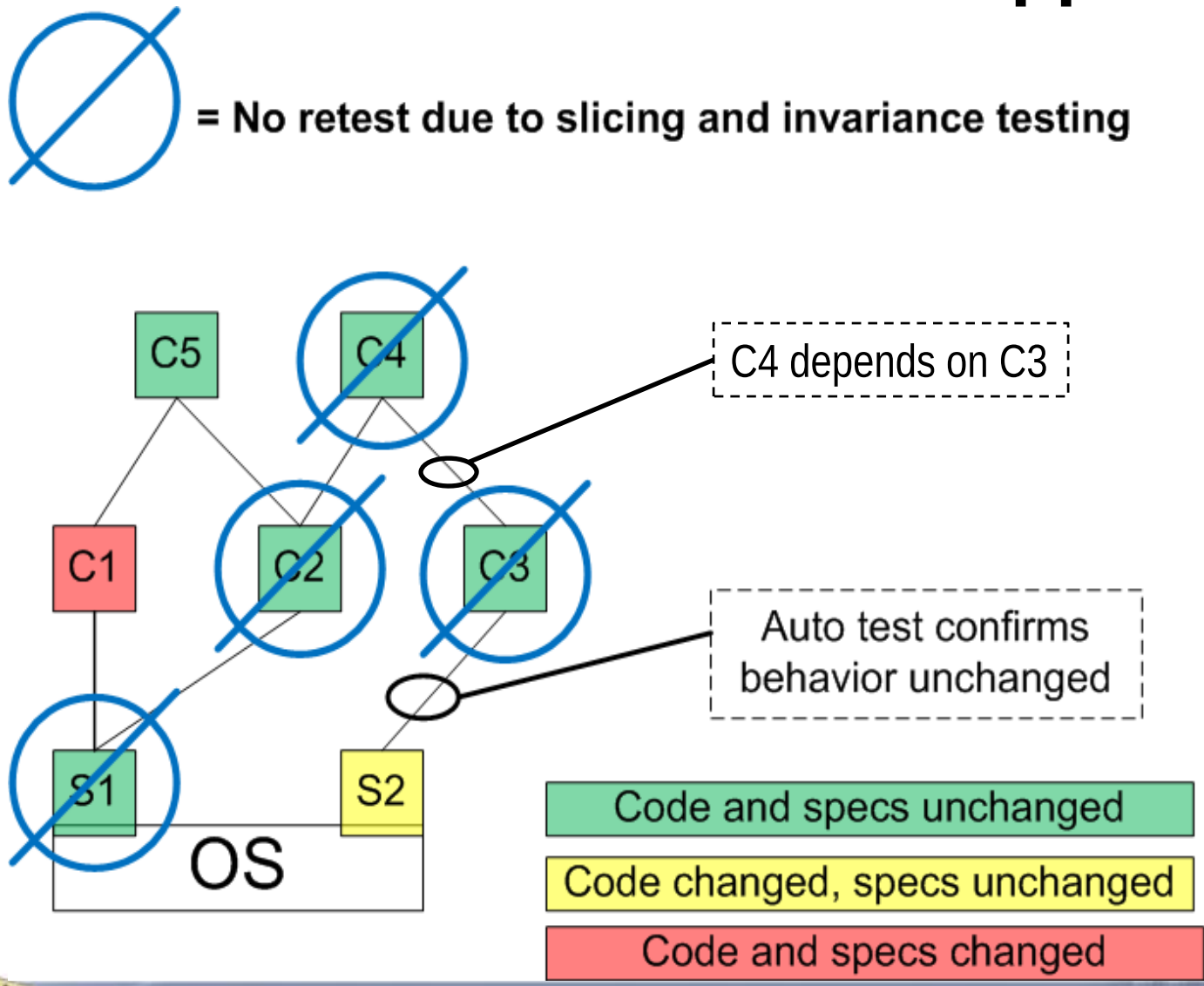
- **A multi-faceted strategy for developing joint interoperable systems that adapt and exploit open system design principles and architectures**

- **OSA Objectives:**
  - Provide more opportunities for completion and innovation
  - **Rapidly field affordable, interoperable systems**
  - **Minimize total ownership cost**
  - Maximize total system performance
  - **Field systems that are easily developed and upgradable**
  - Achieve component software reuse

# Problems and Proposed Solutions

- **New Testing Technologies, Processes & Policies are Needed**
  - Safely Reduce Testing Required (2007-2014)
  - Make testing more effective
    - Risk-based testing (2012-2014)
    - Safe test result reuse (Berzins, 2009)
  - Reduce System Integration problems
    - Architecture QA aimed at system interference (2014)
- **Procedures, practices and guidance are being developed**
  - Open Systems Architecture Technical Reference Frameworks

# Test Avoidance Approach



# Program Slicing

- Program slicing is a kind of automated dependency analysis
  - Same slice implies same behavior
  - Can be computed for large programs
  - Depends on the source code, language specific
  - Some tools exist, but are not in widespread use
  - No tools spanning boundaries between languages (yet)

- Slicing tools must handle the full programming language(s) correctly to support safe reduction of testing.

# Test Reduction Process (1)

- Check that the slice of each service is the same in both versions (automated)
- Check that the requirements and workload of each service are the same in both versions
- Must recheck timing and resource constraints
- Must certify absence of memory corrupting bugs
  - Popular tools exist: Valgrind, Insure++, Coverity, etc.
- Must ensure absence of runtime code modifications due to cyber attacks or physical faults
  - Cannot be detected by testing because modifications are not present in software versions under test
  - Need runtime certification
    - Can be done using cryptographic signatures (Berzins, 2009)

# Test Reduction Process (2)

- The test reduction process in the previous slide is for new releases with the same operating environment.
  - This is a significant constraint because reliability depends strongly on operating environment
  - The same system can have 0% reliability in one environment and 100% in another

- Components reused in different contexts need a different approach
  - Can reuse some previous test results and focus new tests on unexplored parts via differences in operational profiles
    - See (Berzins 2009) for details.
  - Risk-based testing can determine number of test cases needed

# Tool Evaluation

- Two existing Slicing Tools were evaluated for supporting safe test reduction

- Neither was adequate in its current state
  - Incorrect outputs in some cases
  - Both tailored for manual program debugging, graphical output on display screens only
  - No file output or API for integration with other tools
    - Post-processing is needed for large scale analysis

# Risk Based Testing

1. Whole-system operational risk analysis identify potential mishaps / mission failures
2. Identify which software service failures would lead to identified mishaps
3. Use slicing to identify which software modules affect the critical services
4. Associate maximum risk level of affected services with each software module (2012)
5. Set number of test cases using risk level (2008)

# Drone Risk Case Study

(a) Indoor (b) Outdoor

Parrot AR.Drone 2.0

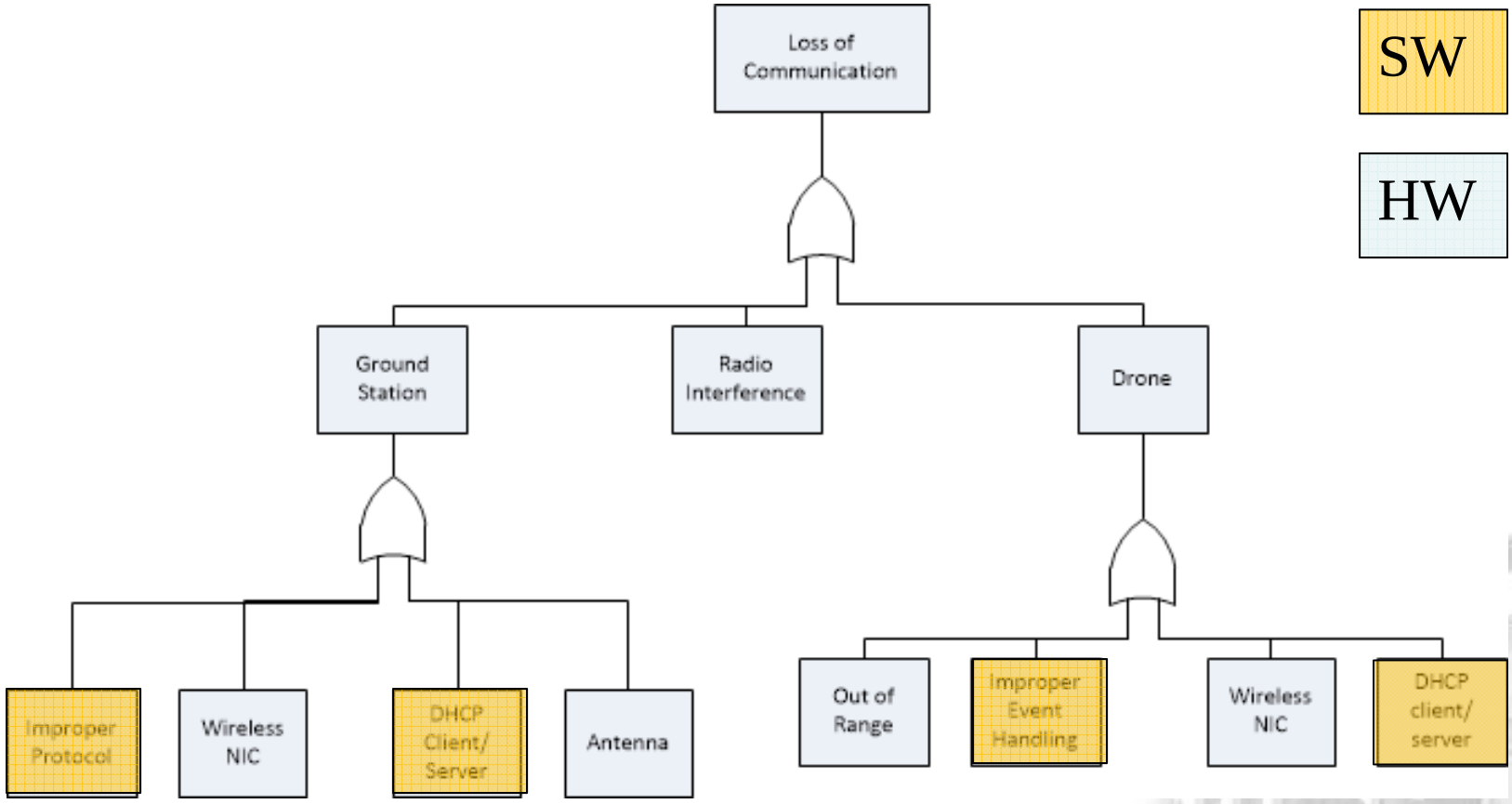
- Commercially available quad-rotor
- Many advanced features such as image recognition
- The Software Development Kit (SDK) is open source and not subject to restrictions

# Drone Hazards

- Hazards that were identified for analysis:
  - Loss of communication
  - Loss of propulsion
  - Damage due to environmental factors
  - Loss of battery power
  - Loss of situational awareness
- Due to schedule limits, the case study focused on damage to the platform, did not include possible damage to surroundings.

# Drone Risk Analysis

- Fault Tree analysis for loss of communication



# Case Study Severity Matrix

| | |
|---|---|
| **Catastrophic** | **Could result in irretrievable loss of aircraft, human life, damages exceeding $1000, or irreversible severe environmental damage that violates law or regulation.** |
| **Critical** | Permanent damage to retrievable aircraft requiring complete replacement, permanent partial disability, loss exceeding $1000 or reversible environmental damage causing a violation of law or regulation or indirectly causes mission failure. |
| **Marginal** | Major reparable damage to aircraft requiring replacement of expensive parts with total repair cost exceeding $150, non-permanent injury or mitigate-able environmental damage causing a violation of law or regulation. |
| **Negligible** | Minor reparable damage to aircraft requiring replacement of cheap parts with total repair not exceeding $150, or minimal environmental damage not violating law or regulation. |

# Case Study Probability Matrix

| | |
|---|---|
| **Frequent** | Expected to occur multiple times in the operation of the aircraft. |
| **Probable** | Will be expected to occur at least once in the operation of the aircraft. |
| **Occasional** | Will be expected to occur at least once after several operations of the aircraft. |
| **Remote** | Unlikely, but can be expected to occur at least once in the life of the aircraft. |
| **Improbable** | Highly unlikely to occur, but still possible to occur at least once in the life of the aircraft. |

# Case Study Risk Analysis Summary

| Hazard | System Risk |
|---|---|
| Loss of Communication | High |
| Loss of Situational Awareness | High |
| Loss of Propulsion | Serious |
| Loss of Battery Power | Medium |
| Environmental Damage | Medium |

Software design changes were suggested to mitigate the highest risks.

The slicing analysis could not be completed due to tool deficiencies.

# Integration and Architecture Faults

- Hypothesis: many system integration problems are due to architecture faults and imperfections in test and evaluation.
  - Examples of integration problems due to architecture faults are on following slides.
  - Testing imperfection example:
    - Code faults in which components fail to conform to architecture standards are missed by test cases.
    - When two components are connected, one triggers such a fault in the other, by exercising an untested situation.
    - Incidence can be reduced by automated statistical testing, with enough test cases for high confidence.

# Architecture Fault Example (1)

- Kitchen plan calls out a Miele microwave oven and an electric outlet.
  1. Electrical contractor installs a 110 volt outlet.
  2. Oven delivered, installation guide requires a 220 volt power supply, installation fails.

- Architecture left out constraints needed to ensure the subsystems will work together.
  - In this case: power supply voltage.

# Architecture Fault Example (2)

- Laundry plan calls out an outlet, water supply, and drain (washer), an outlet, gas supply, and air vent (drier), and a big window on top.
  1. Plumber installs the pipes below the structural members supporting the window.
  2. Electrical contractor finds space for the electrical outlets completely obstructed by the pipes.
- Architecture left out constraints to deconflict resource requirements for the subsystems.
  - In this case: volumes of physical space.

# Recommendations

- Architecture descriptions should be a required deliverable in all contracts for systems with open architectures.
- These should be required to pass design reviews specifically targeted at preventing system integration problems
- System designs & implementations should be checked for conformance to the architecture.
- Preventing unauthorized changes to code should be a requirement for any OSA

# Conclusions

- Program Slicing has potential to reduce the cost of regression testing of new releases.
- Currently available slicing tools need improvement to do this in practice.
- System-level risk analysis can determine the amount of testing needed for software.
- Architecture faults can lead to system integration problems.
- QA procedures aimed at architecture faults should be part of new OSA processes.

# Thank you