Certifying Tools for Test Reduction in Open Architecture

Valdis Berzins

Naval Postgraduate School

U.S. Navy Open Architecture

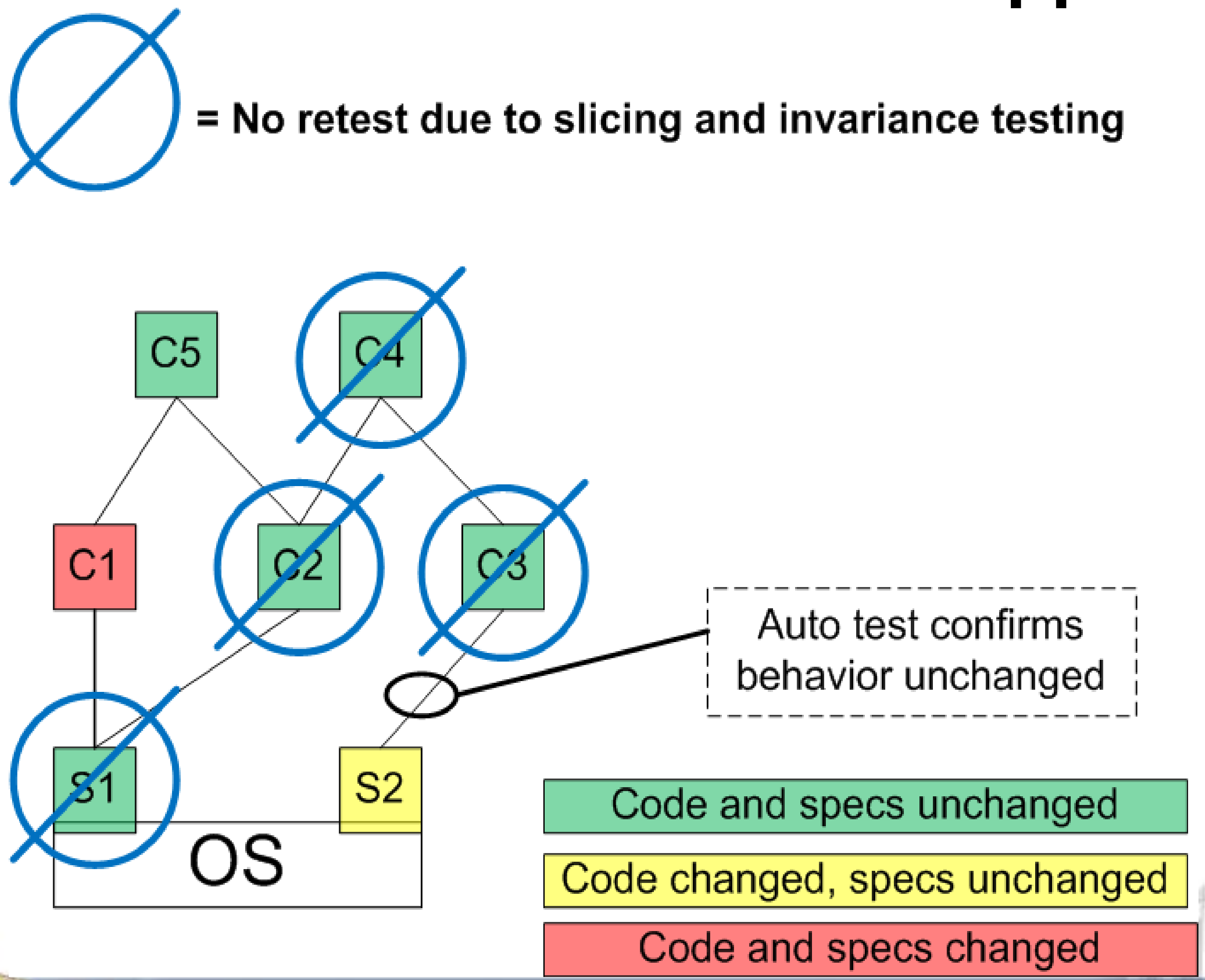
- **A multi-faceted strategy for developing joint interoperable systems that adapt and exploit open system design principles and architectures**

- **OA Principles, processes, and best practices:**
  - 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

Problem and Proposed Solution

- **Traditional U.S. Navy Software T&E practices will limit many benefits of OA**
  - It is virtually impossible to field frequent and rapid configuration changes with current approaches
- **New Testing Technologies, Processes & Policies are Needed**
  - Safely Reduce Testing Required (2007-2012)
  - Make testing more effective
    - Risk-based testing (2012), safe test result reuse (Berzins, 2009)
  - Transition from Manual Testing to Profile-Based Automated Statistical Testing (Berzins, 2010)
  - Dependency-based acquisition (2012)

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

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

Test Reduction Process

- 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 test loads
  - Need runtime certification
    - Can be done using cryptographic signatures (Berzins, 2009)

The Current Problem

To Evaluate the Suitability of
COTS Slicing Tools
for Supporting Safe Test Reduction

Current Research Objectives

1. To conduct experimental assessments and compare the suitability of the available COTS program slicing tools for safe reduction of testing effort.
2. To identify the most adequate slicing tools among the evaluated ones.
3. To determine the suitability of available COTS program slicing tools for practical SW test reduction.
4. To explore additional benefits of dependency analysis

Requirements for Slicing Tools

1. Must satisfy the behavior invariance property:
   - If the original program terminates cleanly, the slices must terminate cleanly and produce the same result as the original program for all observable values specified by the slicing criterion.
2. Must support comparison or output of computed slices
3. Must support modeling of external dependencies

Examples of Dependencies

```
1 int bar(int k) {
2 int v;
3 if (k == 0)
4 v = 1;
5 else
6 v = 2;
7 return v;
8 }
```

```
1 int foo(int k) {
2 Pointer v, u;
3 v = new Pointer();
4 u = v;
5 if (k == 0)
6 v.o = 1;
7 else
8 v.o = 2;
9 u.o = 4;
10 return v.o;
11 }
```

Legend

- Control Dependency
- Data Dependency
- Pointer Aliasing Dependency

Examples of Parallel Dependencies

```
class Account {
private float amount = 0;

public synchronized float withdraw(float x) {
while (amount − x < 0) {
try {wait ();} catch (Exception e) { }
}
amount = amount − x;
return amount;
}
public synchronized float deposit(float x) {
amount = amount + x;
notifyAll ();
return amount;
}
}
```

```
class Worker implements Runnable {
private Account save;
private float amount;
public Worker(Account account, float a) {
save = account;
amount = a;}
public void run() {
save.deposit(amount);
}
}
```

```
class Spouse implements Runnable {
private Account save;
private float amount;
public Spouse(Account account, float a) {
save = account;
amount = a;}
public void run() {
save.withdraw(amount);
(new Account()).deposit(10);
}
}
```

```
class Home {
public static void main(String[] s) {
Account savings = new Account();
Runnable worker = new Worker(savings, 90);
Runnable spouse = new Spouse(savings, 10);
new Thread(worker).start();
new Thread(spouse).start();
}
}
```

Slicing Example

Resolution of slices computed by Kaveri

(a)

```
1  public class Test {
2  public static void main(String[] args) {
3  point p1 = new point();
4  point p2 = new point();
5  p1.a = 1;
6  p2.a = 2; //should not be relevant
7  System.out.println("irrelevant1");
8  System.out.println("P1: a= "+p1.a);
9  System.out.println("irrelevant2");
10 System.out.println("P2: a= "+p2.a);
11 }
12 }
```

Difference

(b)

```
1  public class Test {
2  public static void main(String[] args) {
3  point p1 = new point();
4  point p2 = p1;
5  p1.a = 1;
6  p2.a = 2; //should be relevant
7  System.out.println("irrelevant1");
8  System.out.println("P1: a= "+p1.a);
9  System.out.println("irrelevant2");
10 System.out.println("P2: a= "+p2.a);
11 }
12 }
```

Using slicing criterion {8, p1.a} for both (a) and (b)

Legend:

- Partially Relevant Slice
- 100% Relevant Slice

Project Status

- Experimental assessment is in progress and not yet complete.
  - The team is currently instrumenting the tools and developing additional test cases.

- Developed the initial framework for two additional uses of dependency analysis:
  - Risk based testing
  - Risk based acquisition

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
5. Set number of test cases using risk level

Current Policy for Mishap Risk Assessment

FREQUENCY OF OCCURRENCE	MISHAP SEVERITY CATEGORIES			
	1 CATASTROPHIC	2 CRITICAL	3 MARGINAL	4 NEGLIGIBLE
A – FREQUENT P ≥ 10%	1A	2A	3A	4A
B – PROBABLE 10% > P ≥ 1%	1B	2B	3B	4B
C – OCCASIONAL 1% > P ≥ 0.1%	1C	2C	3C	4C
D – REMOTE .1% > P ≥ 0.0001%	1D	2D	3D	4D
E – IMPROBABLE 0.0001% > P	1E	2E	3E	4E

Cells:	Risk Level & Acceptance Authority:
1A, 1B, 1C, 2A, 2B:	**HIGH** – ASN (RDA)
1D, 2C, 3A, 3B:	**SERIOUS** - PEO-IWS
1E, 2D, 2E, 3C, 3D, 3E, 4A, 4B:	**MEDIUM** –PEO-IWS 3
4C, 4D, 4E:	**LOW** – PEO-IWS 3

P: Probability of occurrence in the lifetime of an individual system, ranges taken from MIL_STD-882D

Risk Based Acquisition

1. Identify missions and scenarios that systems must support
2. Assign priorities to missions / scenarios based on impact of success or failure
3. Use dependency analysis to identify which system components affect mission success
4. Associate maximum priority of affected missions / scenarios with each component
5. Allocate funding per priority level, regardless of which program offices are responsible.

Example

Mission Group Priorities		
Mission Bundle	Priority	Members
Bundle 1	High	M1, M2
Bundle 2	Medium	M1, M3

Inherited Priorities for Individual Missions	
M1	High
M2	High
M3	Medium

Priorities of different bundles must be different

Assumptions

1. It is less contentious to prioritize missions and scenarios than system components
2. In the absence of cross-cutting budget authority, a principled basis for cross-cutting allocation is needed to reach agreement.
3. As more components are shared across platforms, such issues will gain importance.

Conclusion

- For systems with long lifetimes, regression testing is a major cost component in each new release, including periodic technology upgrades.
- Program Slicing has the potential to reduce the time and cost of the regression testing that is necessary to ensure the safety and effectiveness of each new release.
- Preliminary evaluation criteria for slicing tools in the context of their ability to achieve safe reduction of regression testing have been developed.

Thank you