Acquisition Research Program:
Creating Synergy for Informed Change

Test Reduction in Open Architecture via Dependency Analysis

Valdis Berzins,
Peter Lim and Mohsen Ben Kahia

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**

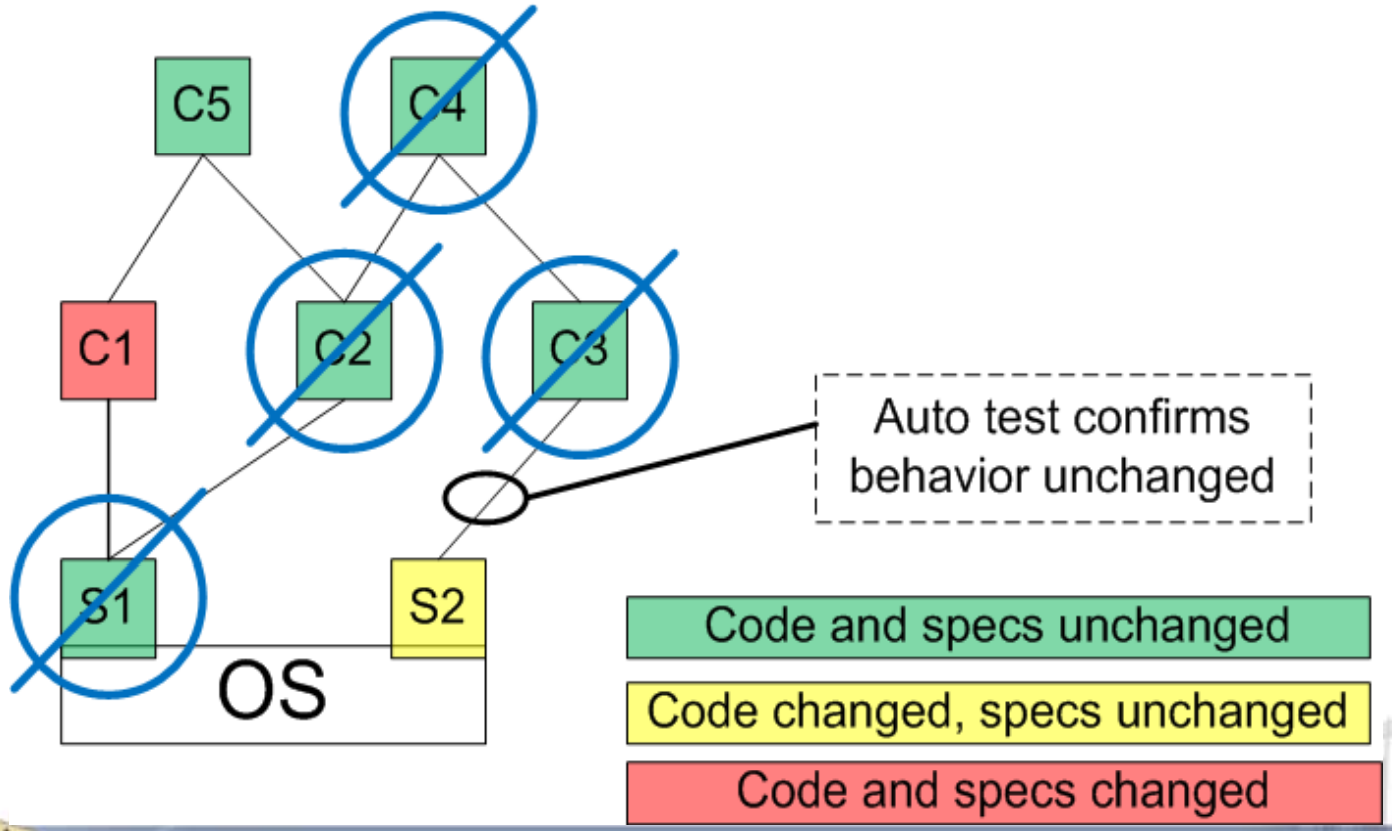
- **OA Principles, processes, and best practices:**
  - Provide more opportunities for completion and innovation
  - Rapidly field affordable, interoperable systems
  - Minimize total ownership cost
  - Maximize total system performance
  - Field systems that are easily developed and upgradable
  - Achieve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

Problem and Proposed Solution

- **Traditional U.S. Navy Software T&E practices will limit many benefits of OA**
  - It will be virtually impossible to field frequent and rapid configuration changes
- **New Testing Technologies, Processes & Policies are Needed**
  - Safely Reduce Amount of Testing Required
  - Transition from Manual Testing to Profile-Based Automated Statistical Testing (Berzins, 2010)

Test Avoidance Approach

= No retest due to slicing and invariance testing



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
- Slicing tools must handle the full programming language correctly

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
  - Tools exist: Valgrind, Insure++, Coverity, etc.
- Must ensure absence of runtime code modifications due to cyber attacks
  - Cannot be detected by testing because modifications are not present in test loads
  - Need runtime checking, 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 provide criteria for evaluating and applying program slicing tools to safely reduce re-testing of SW components in the new SW releases.
2. To conduct experimental assessments and compare the suitability of the available COTS program slicing tools for safe reduction of testing effort.
3. To identify the most adequate slicing tools among the evaluated ones.
4. To determine the suitability of available COTS program slicing tools for practical SW test reduction.

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. All slices must be executable if the original program is.
   - Programs that fail to terminate or terminate abnormally are considered to be executable in our context.

Language features that present Slicing Challenges

- Object Oriented programs
  - Classes and their instances
  - Objects
  - Inheritance
  - Polymorphism
  - Dynamic binding
- Pointers

  Aliasing ➔ safe approximations are necessary
- Concurrent programs
  - Inter-process synchronization among multiple control flows
  - Inter-process communication among multiple data flows
- External Calls

Dependencies Relevant to Slicing

- Data Dependencies
- Control Dependencies
- Parallel Dependencies
  - Selection Dependencies
  - Synchronization Dependencies
  - Internal-Communication Dependencies
- External Dependencies
  - System calls
  - External Libraries
  - Databases
  - External application level services

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

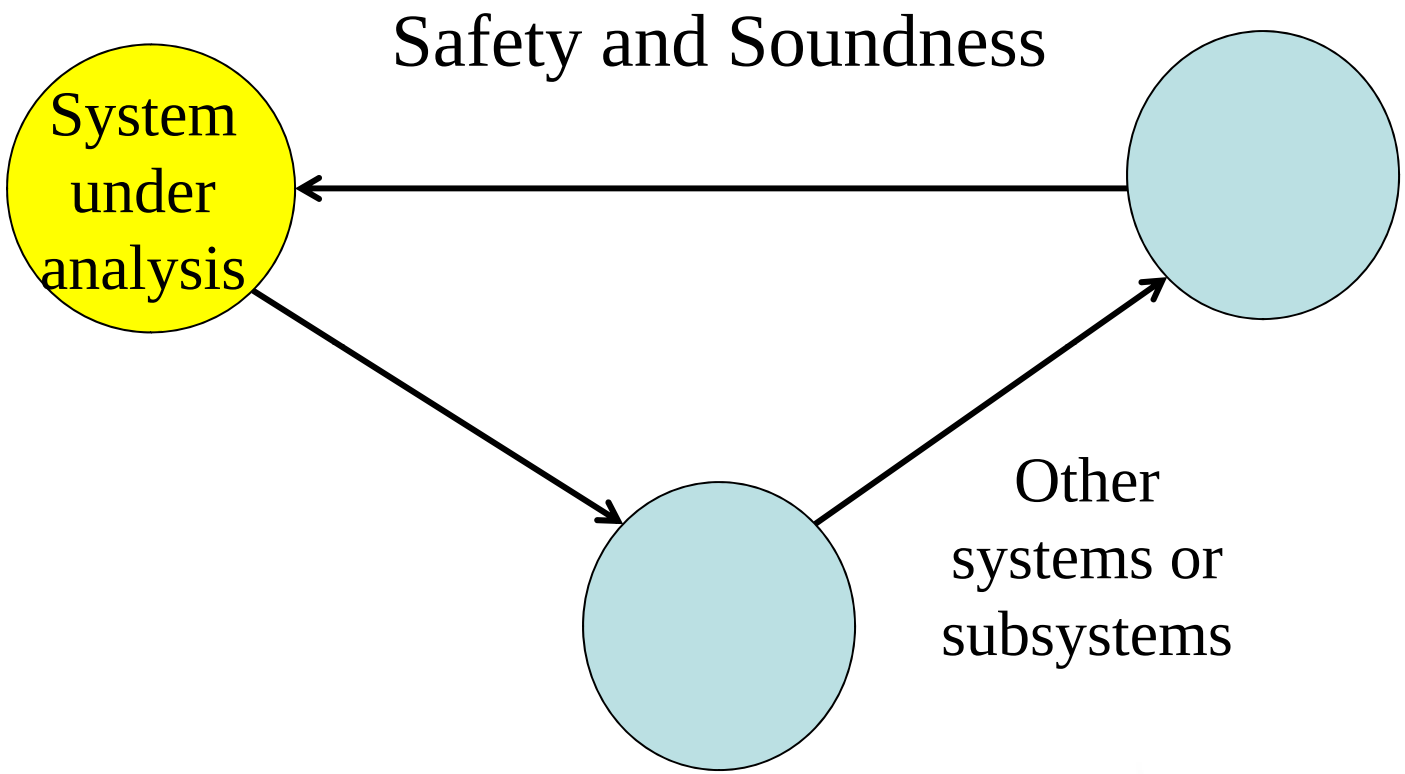
```
class Account {
private float amount = 0;

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);
}
}
```

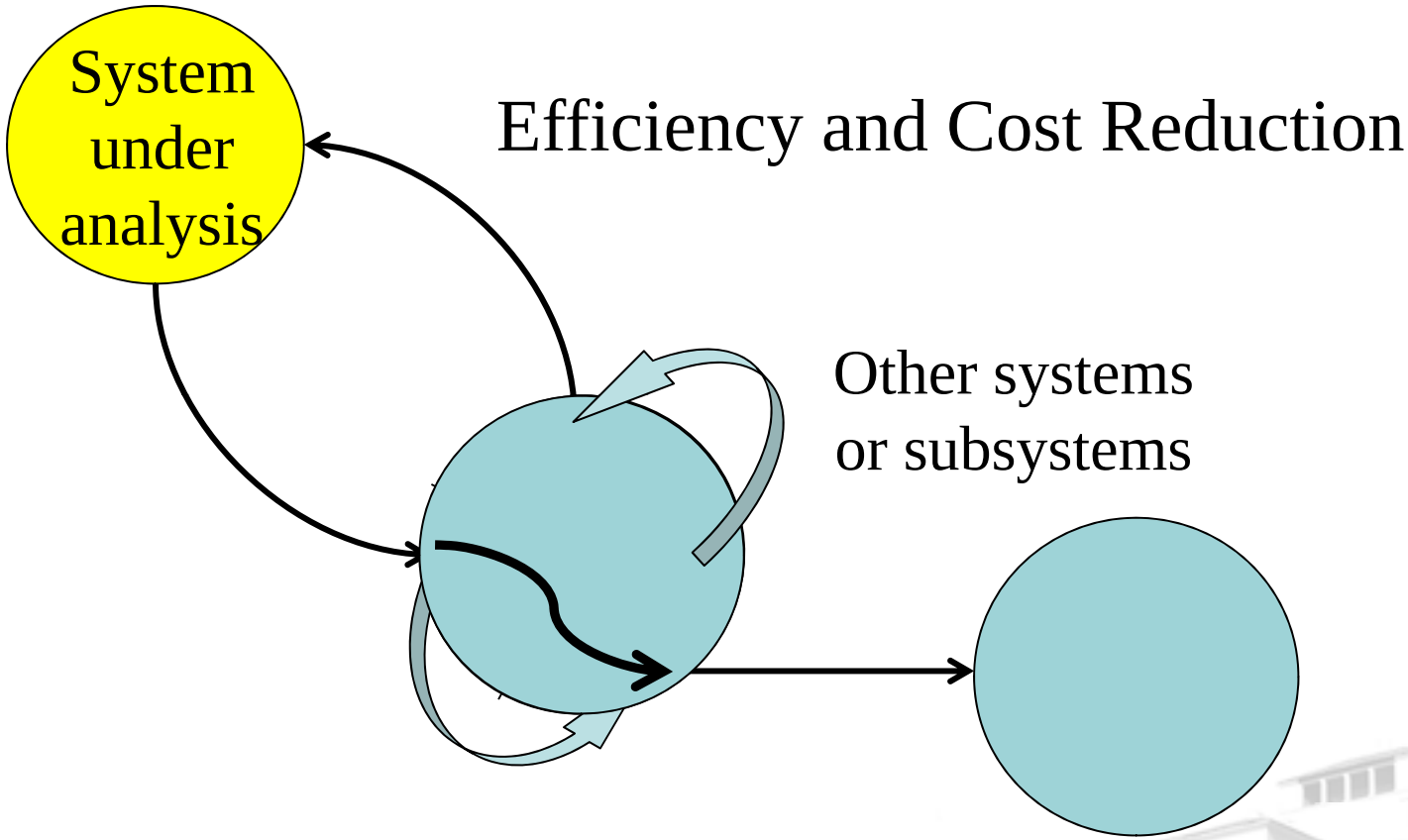
```
class Home {
public static void main(String[] s) {
Account savings = new Account();
Runnable worker = new Worker(savings, 90);
Runnable spouse = new Spouse(savings, 10);
new Thread(worker).start();
new Thread(spouse).start();
}
}
```

External dependencies



➔ Disregarding External dependencies may violate the behavior invariance property of the slice

External dependencies cont.



➔ The inaccuracy of evaluating external dependencies may result in overestimated slices.

Adequacy Criteria For Slicing Tools

- Programming languages handled
- Behavior of the computed slices
- Size of the computed slices
- Pointers and parameter passing
- Capability for slicer output or slice comparison
- Capability for modeling external components

Slicing Example

Resolution of slices computed by Kaveri

```
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 }
```

(a)

Difference

```
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 }
```

(b)

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

Legend

Partial Relevant Slice

Relevant Slice

Assessment Scope

- Plan to use the following slicing tools in our evaluation:

a) Indus's static slicing tool for Java, developed by Kansas State University and delivered as an Eclipse plug-in under the product name Kaveri.

b) GrammaTech's CodeSurfer static slicing tool for C/C++, formerly developed by Wisconsin Slicing Project.

c) Jslice static and dynamic slicing tool for Java, developed by the National University of Singapore.

Assessment Scope

- Academic slicing tools developed for research purposes that lack documentation and support may not be evaluated:

a) Unravel static slicing tool for C, a prototype tool contracted to the NIST by the US Nuclear Regulatory Commission and the National Communications System. (dated documentation)

b) Oberon Slicing Tool (OST) for Oberon system, developed by the Johannes Kepler University. (Oberon is a modern version of Pascal and not widely use in the defense industry)

Project Status

- The team is currently testing some of the tools and will provide a comprehensive test driven adequacy criteria and test cases in a later publication.
- Experimental assessment is currently in progress and is not yet complete.

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.

Next Steps

- If the result from the tool assessment is positive:
  a) Use the chosen slicing tools to identify possible reductions in regression testing for part of a real system.
  b) Conduct a pilot study to check the safety and effectiveness of the theoretically proposed approach.
- If the result from the tool assessment is not positive:
  a) Identify the candidate tools that are closest to meet the requirements for supporting safe regression test reduction and their current shortcomings.

Thank you