Jazz: A Tool for Demand-Driven Structural Testing

Bruce Childers (childers@cs.pitt.edu)

Jon Misurda, James A. Clause, Juliya L. Reed

Department of Computer Science
University of Pittsburgh
Pittsburgh, Pennsylvania 15260 USA

Mary Lou Soffa

Department of Computer Science
University of Virginia
Charlottesville, Virginia 22904 USA

Structural Software Testing

- Assure quality, robust software
  - Collect coverage information about the program
- Coverage types
  - Node coverage determines basic blocks executed
  - Branch coverage determines edges executed
  - Def-use coverage determines pairs of variable definitions and uses that are executed
- Over multiple inputs until coverage criteria

Current Tools and Approaches

- E.g., JCover, PurifyPlus
- Use static program instrumentation
  - Injected prior to program execution
  - E.g., instrument basic blocks with “hit counter” to indicate when a block is executed
- Limitations
  - Not scalable: Instrumentation remains in program
  - Inflexible: Only certain tests, languages, platforms

Our Approach

- A scalable & flexible framework
  - Automatically apply multiple test strategies
  - Multiple languages and platforms
  - Handle large programs
- Demand-driven structural testing [ICSE'05]
  - **Specification driven:** User written test
  - **Test plans:** Recipe of how & where to test
  - **Path specific:** Instrument only what is needed
  - **Dynamic:** Insert & remove instrumentation

Jazz – A Framework Instance

- Structural testing for Java programs
  - Branch, node, def-use coverage
  - User-written specification, demand-driven testing, result reports
- Implementation
  - Eclipse 3.01 plug-in: Test specification & reporting
  - Jikes RVM: Demand-driven testing
  - Works on all programs runnable by Jikes
  - x86/Linux

Jazz Demo

- Branch coverage of music player
  - Test region is whole program (JOrbis)
  - Test input is a song
- Compared
  - Traditional approach with static instrumentation
  - Demand-driven approach with dynamic instrumentation

Traditional Approach **Demand-driven Approach**

Jazz Demo

- Branch coverage of music player
  - Test region is whole program (JOrbis)
  - Test input is a song
- Compared
  - Traditional approach with static instrumentation
  - Demand-driven approach with dynamic instrumentation

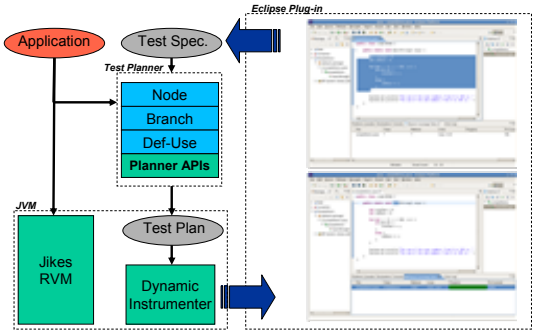
Traditional Approach **Demand-driven Approach**

Jazz Demo

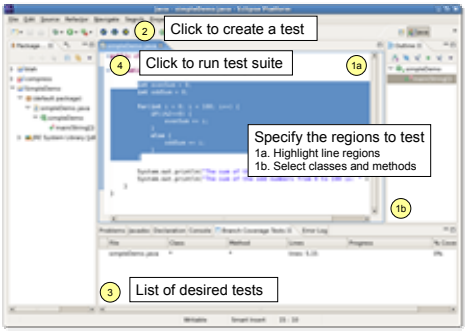
- Branch coverage of music player
  - Test region is whole program (JOrbis)
  - Test input is a song
- Compared
  - Traditional approach with static instrumentation
  - Demand-driven approach with dynamic instrumentation

Traditional Approach **Demand-driven Approach**

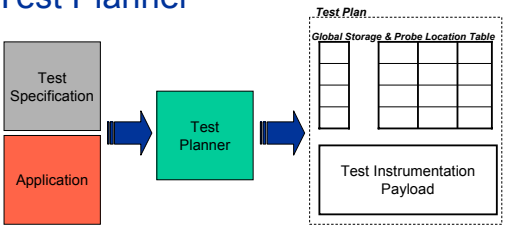
Jazz Tool Flow



Test Specification



Test Planner

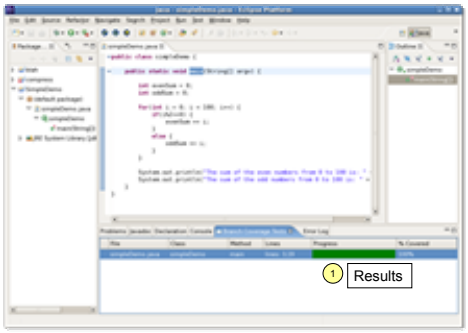


- **Test plan**: Where and how to test program
- **Test storage**, **probe location table**, **instrumentation payload**
- Combine payloads to create custom strategies

Dynamic Instrumentation

- Test plan targets an instrumentation API
- FIST instrumentation engine [WOSS'04]
  - Retargetable & reconfigurable
  - Dynamic insertion & removal of instrumentation
  - Binary level instrumentation (post JIT)
- Uses ***fast breakpoints*** [Kessler]: Replace existing instruction with a jump to instrumentation

Test Results



Branch Coverage Example

- Record which edges are executed
  - Determine (source, sink) edge pairs hit at run-time
  - Source is a branch
  - Sink can be taken & not-taken target of branch
- Within a test region, dynamically instrument along path of execution
  - Insert instrumentation at edge sink blocks
  - Remove instrumentation at edge source as soon as branch covered

Branch Coverage Example

Example program: *simpleDemo.java*

```
public class simpleDemo {
    public static void main(String[] args) {
        int evenSum = 0;
        int oddSum = 0;

        for(int i = 0; i < 100; i++) {
                if(i%2==0) {
                        evenSum += i;
                }
                else {
                        oddSum += i;
                }
        }
        System.out.println("The sum of the even numbers from 0 to 100 is: " + evenSum);
        System.out.println("The sum of the odd numbers from 0 to 100 is: " + oddSum);
    }
}
```

Coverage 100%

Use Jazz GUI to determine test specification: *simpleDemo.testspec*

```
DEFINITIONS {
      NAME: method_main, REGION_D,
      LOCATION: FILE simpleDemo.java {
            CLASS simpleDemo, METHOD main
      }
}
BODY {
      DO BRANCH_TEST ON REGION method_main UNTIL: 100%
}
```

Branch Coverage Example

Probe Loc Tab

Block	Next
1	2,3
2	4
3	4
4	1

Storage

Covered

M, M

M

M

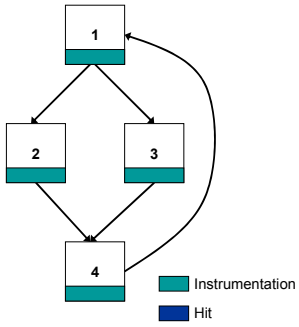
M

Test Payload

Mark edge hit

Insert at next point

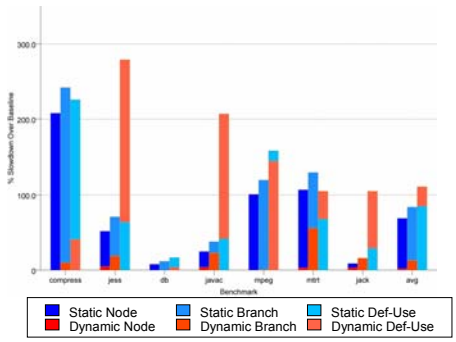
Remove instrumentation



Experiments

- Traditional vs. Jazz (in same implementation)
  - Compared coverage and run-time
- SPECjvm98 benchmarks
  - unloaded 2.4 Ghz Pentium IV, 1GB of memory
  - RedHat Linux 7.3
- Coverage – same reported by both tools
  - Branch coverage: 38.9-58%
  - Node coverage: 75-90.6%
  - Def-use coverage: 66.9-90.5%

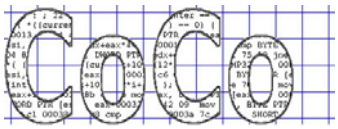
Run-time Performance



Summary

- A new tool (**Jazz**) for structural testing
  - Implements demand-driven approach
  - Test specification in Eclipse IDE
  - Test planner & dynamic instrumentation in Jikes
- Very low overhead
  - E.g., branch coverage tool is 3-4x faster than traditional approaches

For Further Information…



Continuous Compilation

childers@cs.pitt.edu

http://www.cs.pitt.edu/coco