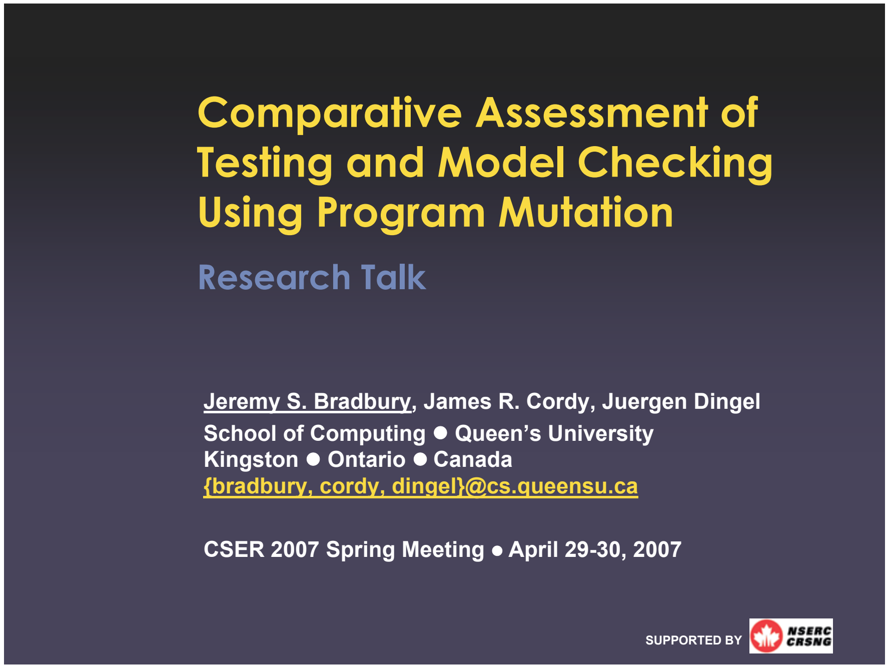


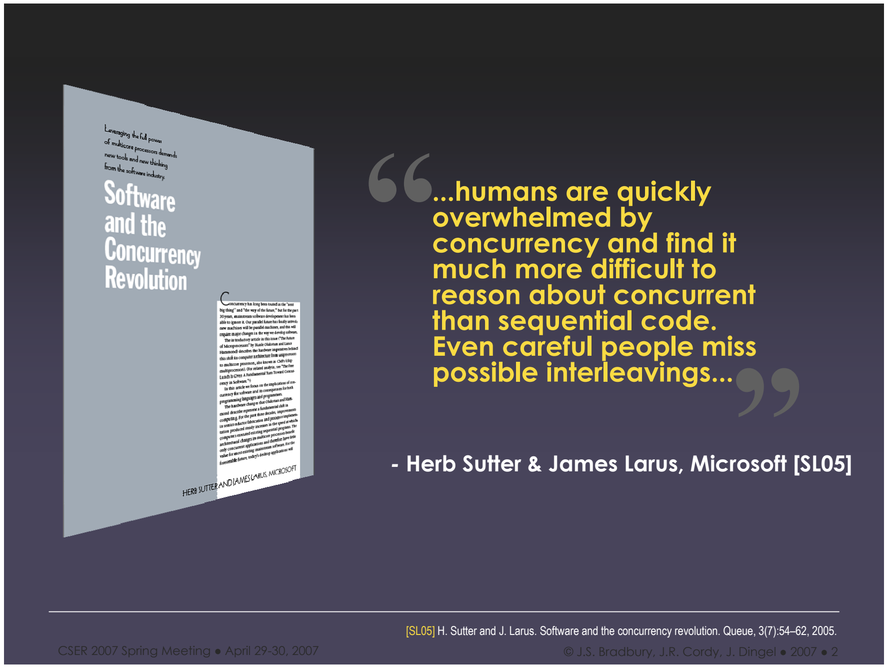
Comparative Assessment of Testing and Model Checking Using Program Mutation

Research Talk

Jeremy S. Bradbury, James R. Cordy, Juergen Dingel
School of Computing ● Queen's University
Kingston ● Ontario ● Canada
{bradbury, cordy, dingel}@cs.queensu.ca

CSER 2007 Spring Meeting ● April 29-30, 2007

SUPPORTED BY NSERC CRSNG



Leveraging the full power of multicore processors demands new tools and new thinking from the software industry.

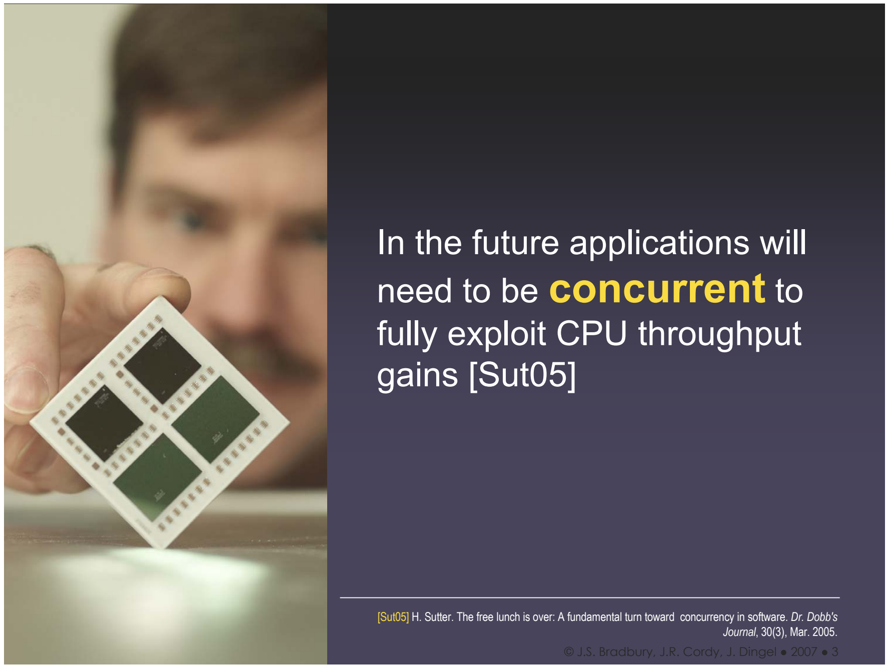
Software and the Concurrency Revolution

HERB SUTTER AND JAMES LARUS, MICROSOFT

> "...humans are quickly overwhelmed by concurrency and find it much more difficult to reason about concurrent than sequential code. Even careful people miss possible interleavings..."

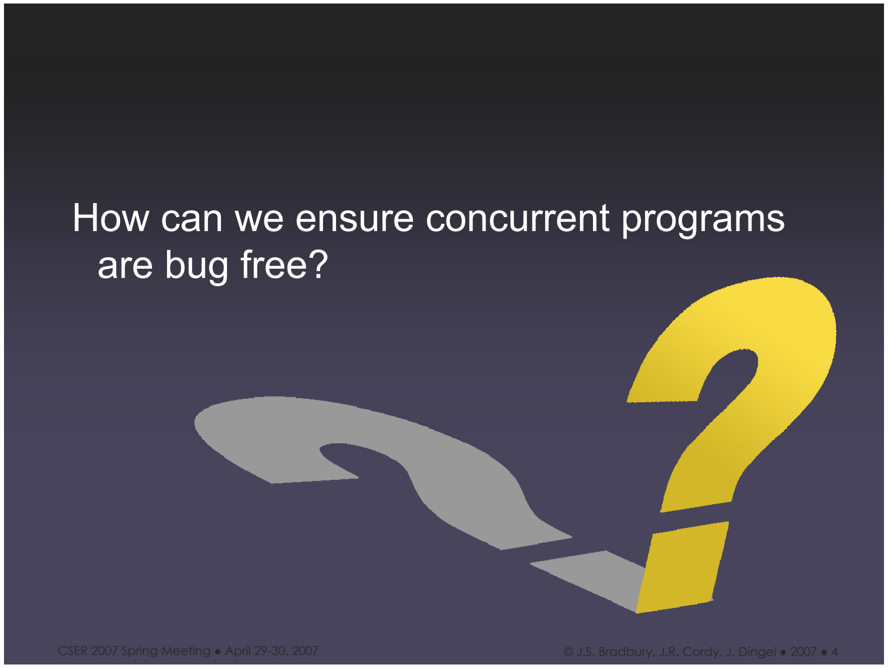
- Herb Sutter & James Larus, Microsoft [SL05]

[SL05] H. Sutter and J. Larus. Software and the concurrency revolution. Queue, 3(7):54–62, 2005.

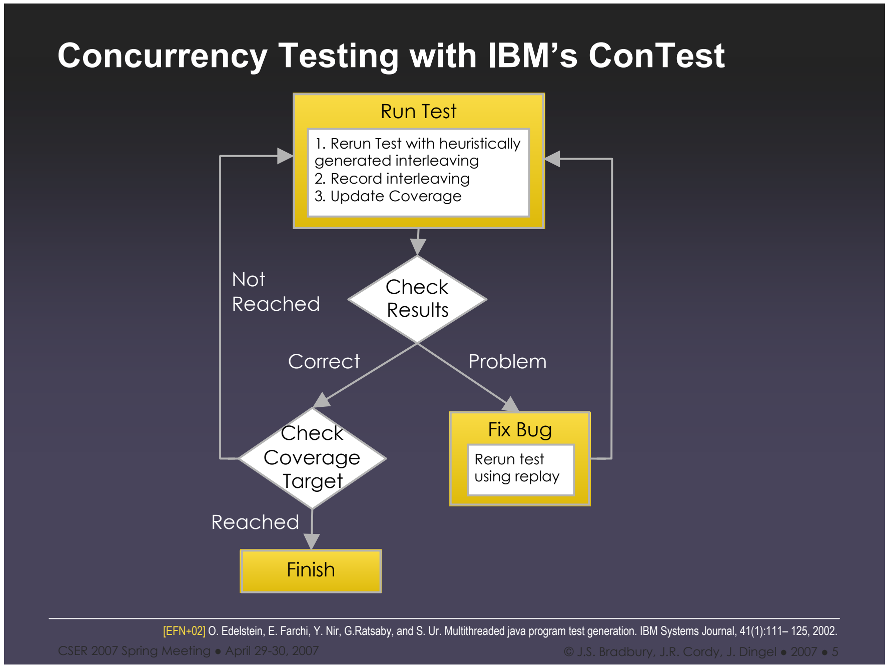


In the future applications will need to be **concurrent** to fully exploit CPU throughput gains [Sut05]

[Sut05] H. Sutter. The free lunch is over: A fundamental turn toward concurrency in software. *Dr. Dobb's Journal*, 30(3), Mar. 2005.



How can we ensure concurrent programs are bug free?



Concurrency Testing with IBM's ConTest

Run Test

1. Rerun Test with heuristically generated interleaving
2. Record interleaving
3. Update Coverage

Check Results

Not Reached

Correct

Problem

Check Coverage Target

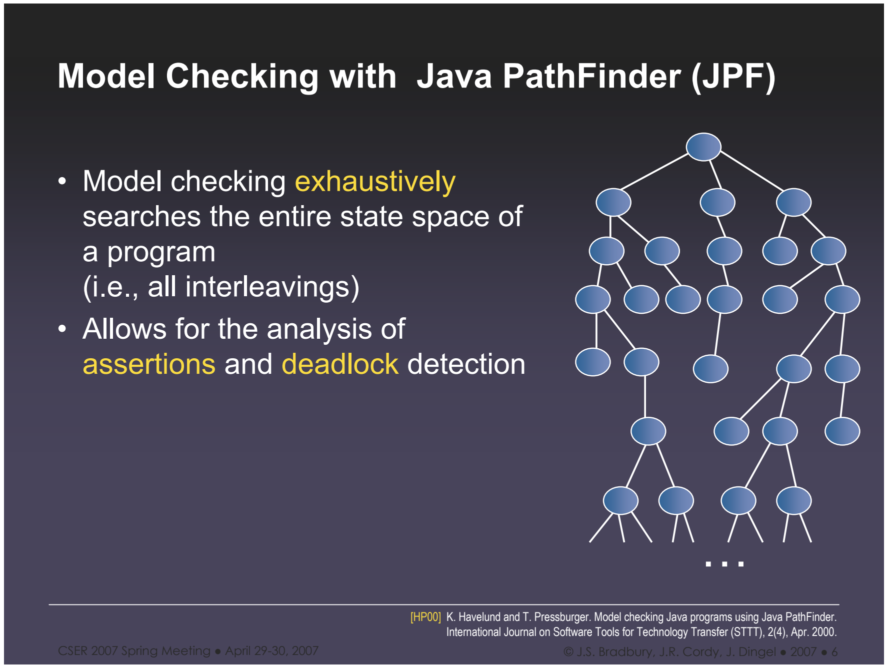
Fix Bug

Rerun test using replay

Reached

Finish

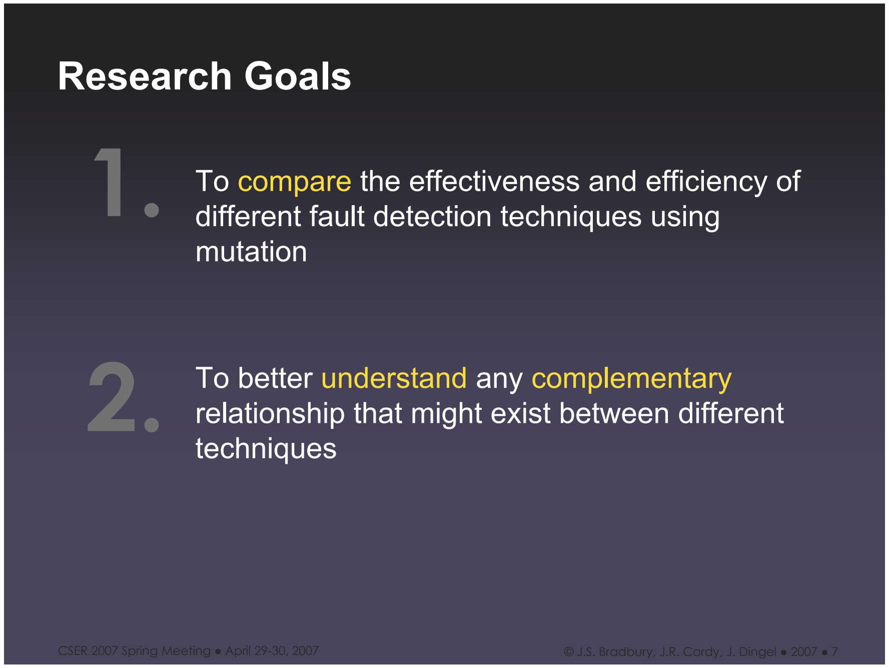
[EFN+02] O. Edelstein, E. Farchi, Y. Nir, G.Ratsaby, and S. Ur. Multithreaded java program test generation. IBM Systems Journal, 41(1):111– 125, 2002.



Model Checking with Java PathFinder (JPF)

- Model checking exhaustively searches the entire state space of a program (i.e., all interleavings)
- Allows for the analysis of assertions and deadlock detection

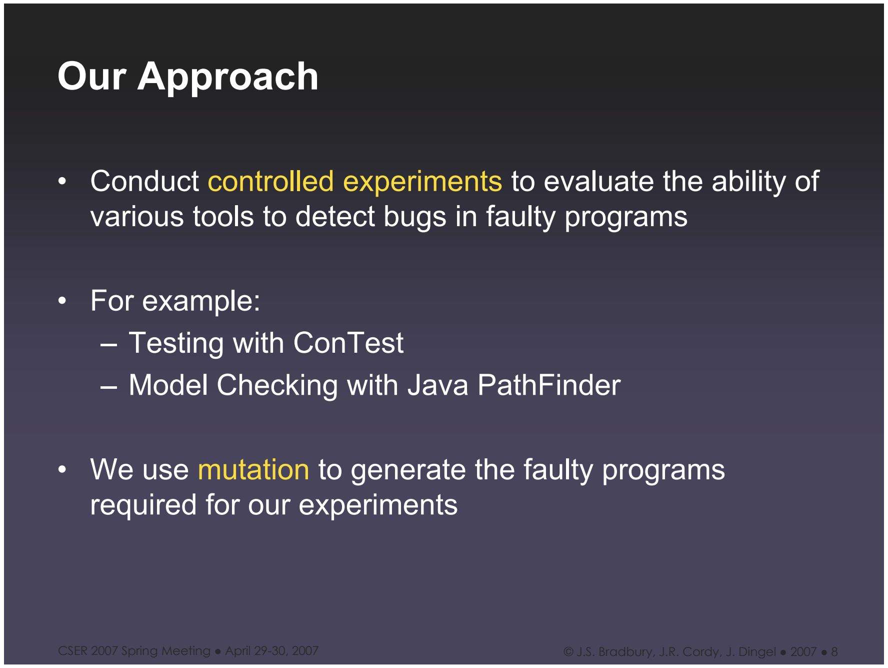
[HP00] K. Havelund and T. Pressburger. Model checking Java programs using Java PathFinder. International Journal on Software Tools for Technology Transfer (STTT), 2(4), Apr. 2000.



Research Goals

1. To compare the effectiveness and efficiency of different fault detection techniques using mutation

2. To better understand any complementary relationship that might exist between different techniques



Our Approach

- Conduct controlled experiments to evaluate the ability of various tools to detect bugs in faulty programs
- For example:
  - Testing with ConTest
  - Model Checking with Java PathFinder
- We use mutation to generate the faulty programs required for our experiments

Our Approach

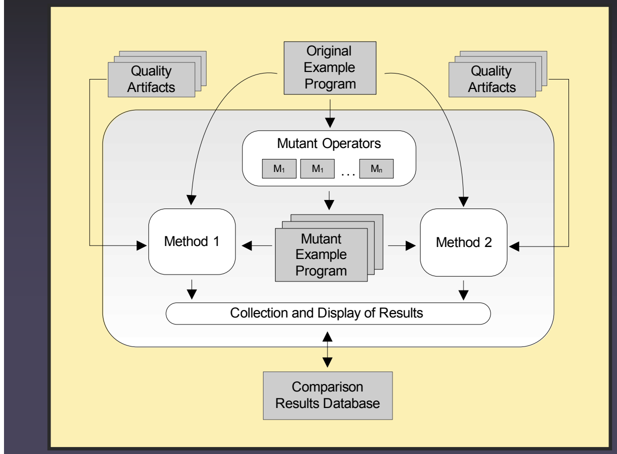
- Mutation [Ham77,DLS78] traditionally used within the sequential testing community
  - –evaluate the effectiveness of test suites
- Mutation relies on *mutation operators (patterns)* to generate faulty versions of the original program called *mutants*

Mutant score of *t* = % of mutants detected (*killed*) by a technique *t* (e.g., testing, model checking)

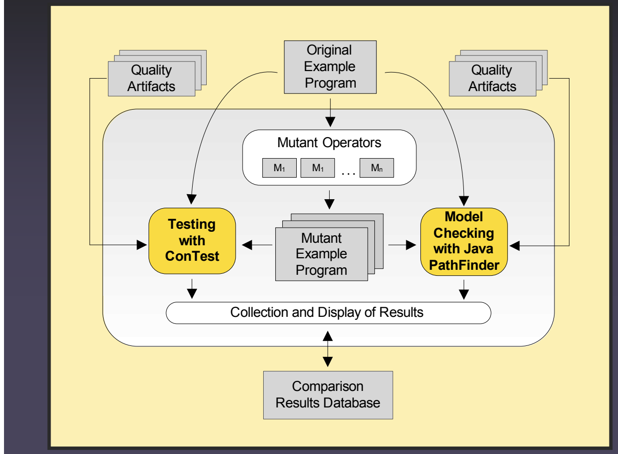
[Ham77] R.G. Hamlet. Testing programs with the aid of a compiler. IEEE Trans. on Soft. Eng., 3(4), Jul. 1977.

[DLS78] R. A. DeMillo, R. J. Lipton, and F. G. Sayward. Hints for test data selection: help for the practicing programmer. IEEE Computer, 11(4):34–41, Apr. 1978.

Research Methods

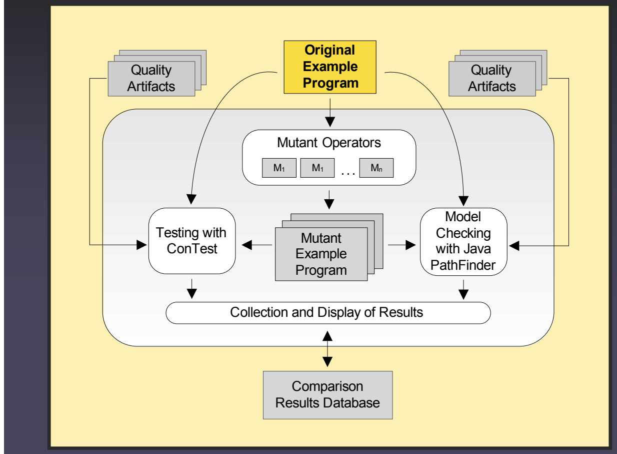


Experimental Setup



Approach Selection

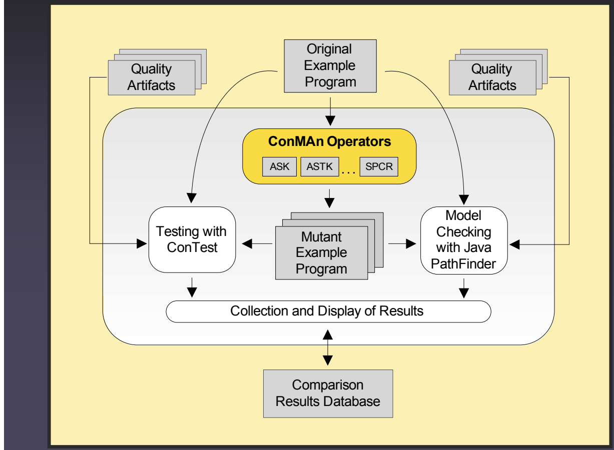
Experimental Setup



Approach Selection

Example Program Selection

Experimental Setup



Approach Selection

Example Program Selection

Mutation Selection

The ConMAn Operators

- **ConMAn** = **Con**currency **M**utation **An**alysis
- What are the ConMAn operators?
  - *"…a comprehensive set of 24 operators for Java that are representative of the kinds of bugs that often occur in concurrent programs."*
  - based on an existing fault model for Java concurrency [FNU03]
- Can be used as a comparative metric

[FNU03] E. Farchi, Y. Nir, and S. Ur. Concurrent bug patterns and how to test them. In *Proc. of IPDPS 2003.*

Example ConMAn Mutation

SKCR – Shrink Critical Region

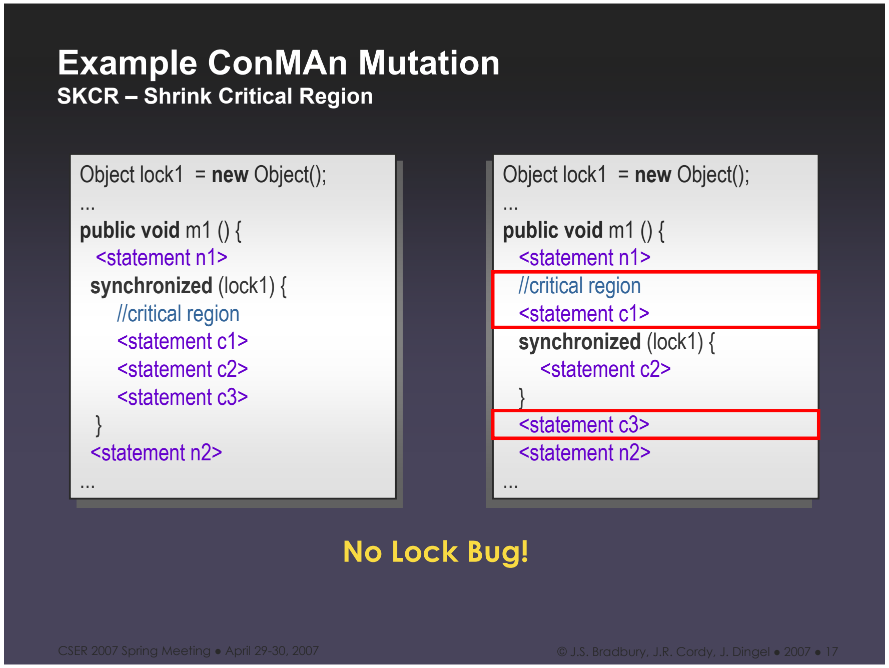
```
Object lock1  = new Object();
...
public void m1 () {
  <statement n1>
 synchronized (lock1) {
     //critical region
     <statement c1>
     <statement c2>
     <statement c3>
  }
 <statement n2>
...
```

Example ConMAn Mutation

SKCR – Shrink Critical Region

```
Object lock1  = new Object();
...
public void m1 () {
  <statement n1>
 synchronized (lock1) {
     //critical region
     <statement c1>
     <statement c2>
     <statement c3>
  }
 <statement n2>
...
```

```
Object lock1  = new Object();
...
public void m1 () {
  <statement n1>
  //critical region
  <statement c1>
  synchronized (lock1) {
     <statement c2>
  }
  <statement c3>
  <statement n2>
...
```



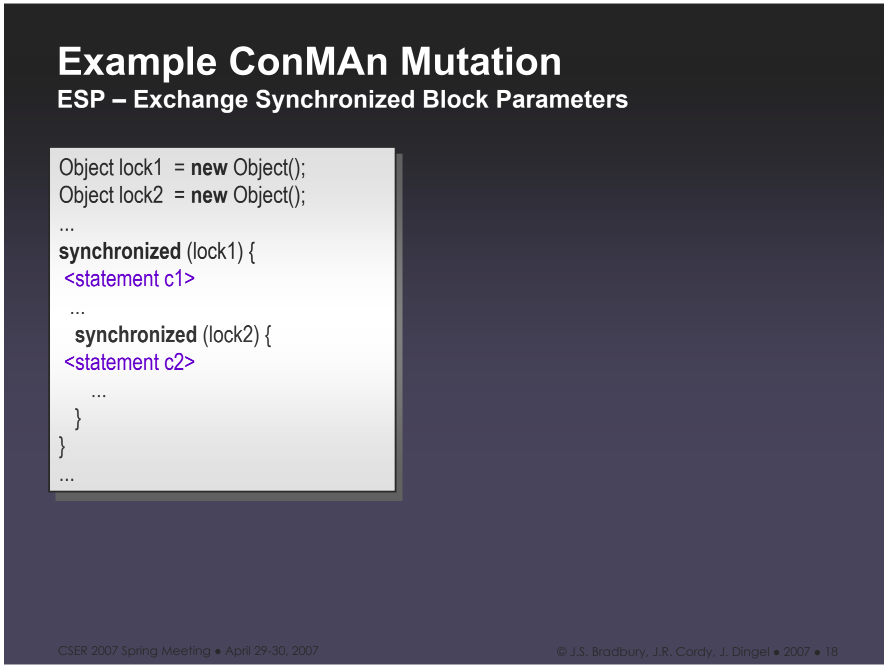
Example ConMAn Mutation

SKCR – Shrink Critical Region

```
Object lock1  = new Object();
...
public void m1 () {
  <statement n1>
  synchronized (lock1) {
     //critical region
     <statement c1>
     <statement c2>
     <statement c3>
  }
  <statement n2>
...
```

```
Object lock1  = new Object();
...
public void m1 () {
  <statement n1>
  //critical region
  <statement c1>
  synchronized (lock1) {
     <statement c2>
  }
  <statement c3>
  <statement n2>
...
```

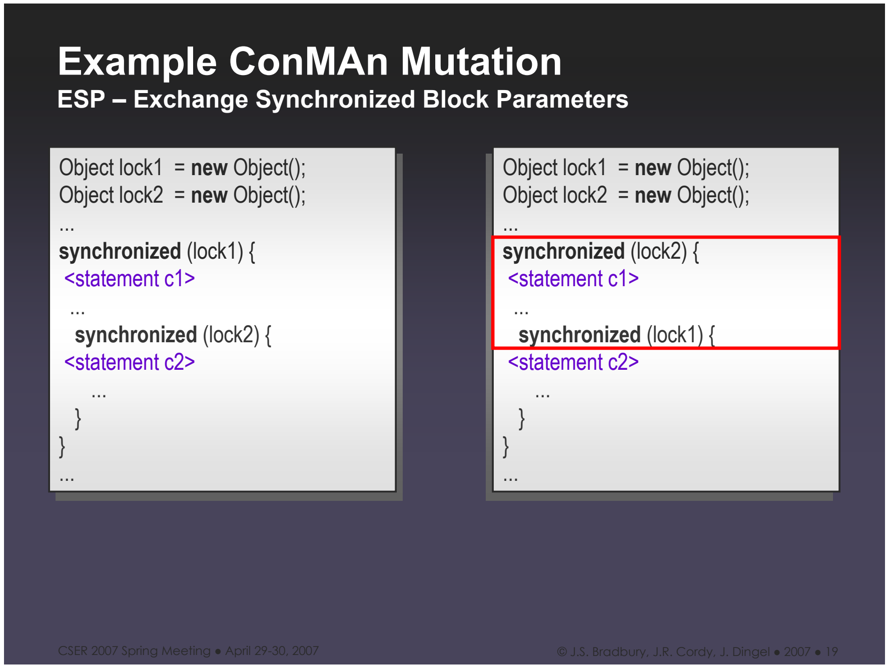
No Lock Bug!



Example ConMAn Mutation

ESP – Exchange Synchronized Block Parameters

```
Object lock1  = new Object();
Object lock2  = new Object();
...
synchronized (lock1) {
 <statement c1>
  ...
  synchronized (lock2) {
 <statement c2>
     ...
  }
}
...
```

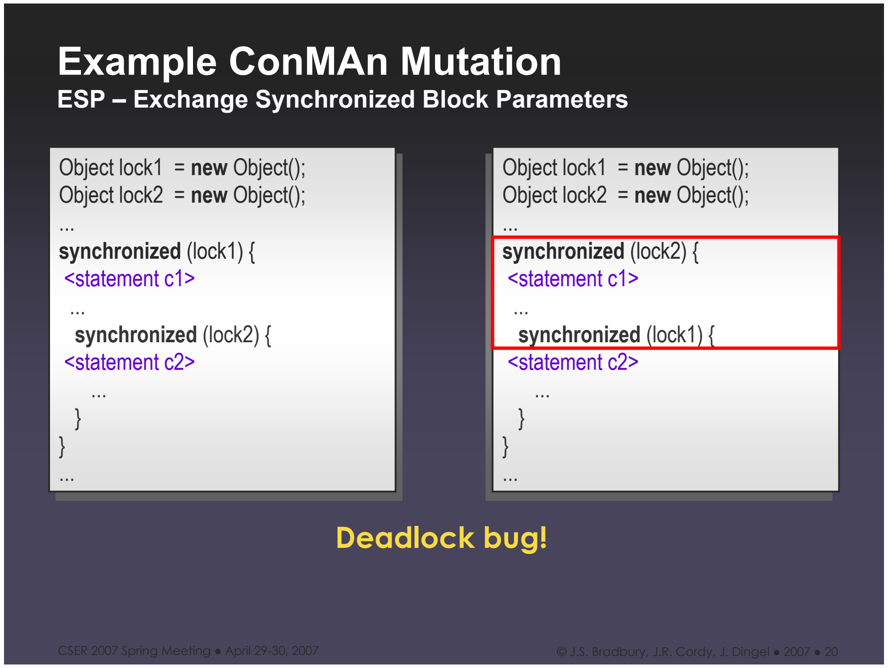


Example ConMAn Mutation

ESP – Exchange Synchronized Block Parameters

```
Object lock1  = new Object();
Object lock2  = new Object();
...
synchronized (lock1) {
 <statement c1>
  ...
  synchronized (lock2) {
 <statement c2>
     ...
  }
}
...
```

```
Object lock1  = new Object();
Object lock2  = new Object();
...
synchronized (lock2) {
 <statement c1>
  ...
  synchronized (lock1) {
 <statement c2>
     ...
  }
}
...
```



Example ConMAn Mutation

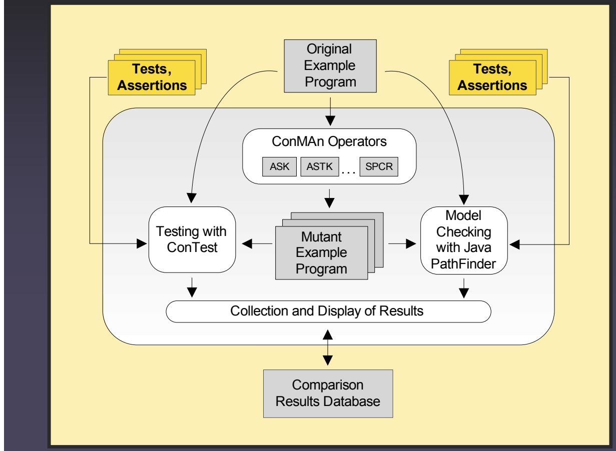
ESP – Exchange Synchronized Block Parameters

```
Object lock1  = new Object();
Object lock2  = new Object();
...
synchronized (lock1) {
 <statement c1>
  ...
  synchronized (lock2) {
 <statement c2>
     ...
  }
}
...
```

```
Object lock1  = new Object();
Object lock2  = new Object();
...
synchronized (lock2) {
 <statement c1>
  ...
  synchronized (lock1) {
 <statement c2>
     ...
  }
}
...
```

Deadlock bug!

Experimental Setup



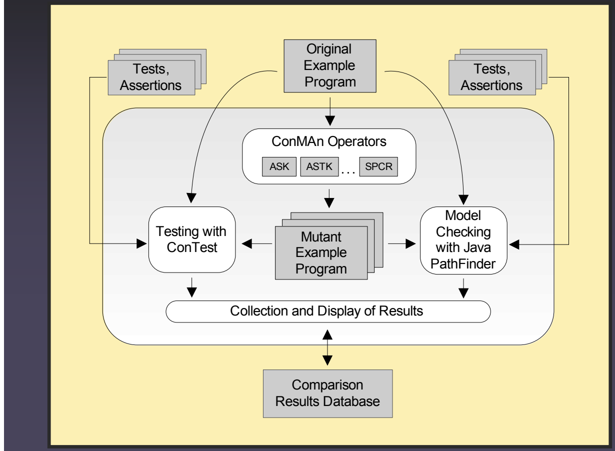
Approach Selection

Example Program Selection

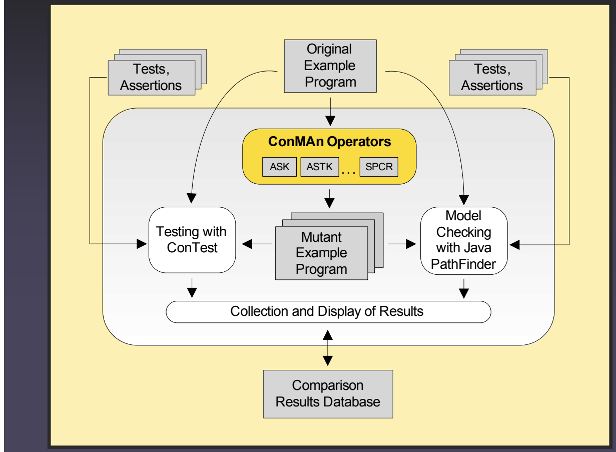
Mutation Selection

Program Artifact Selection

Experimental Procedure

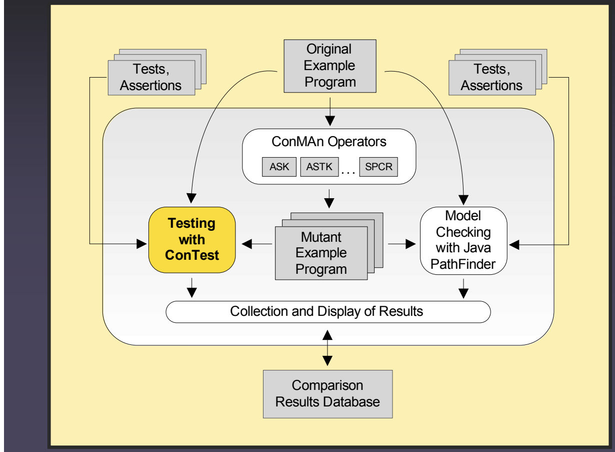


Experimental Procedure



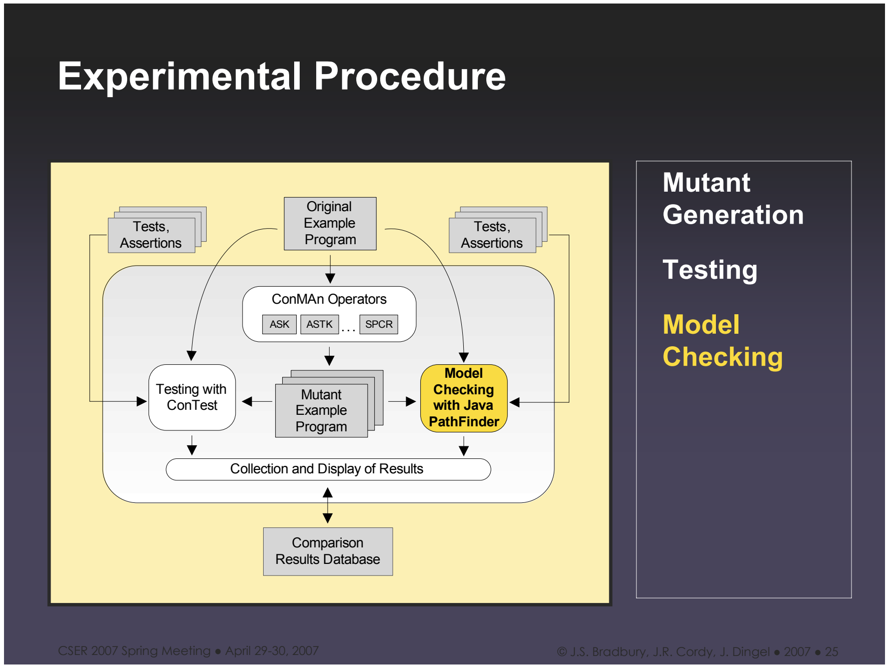
Mutant Generation

Experimental Procedure



Mutant Generation

Testing

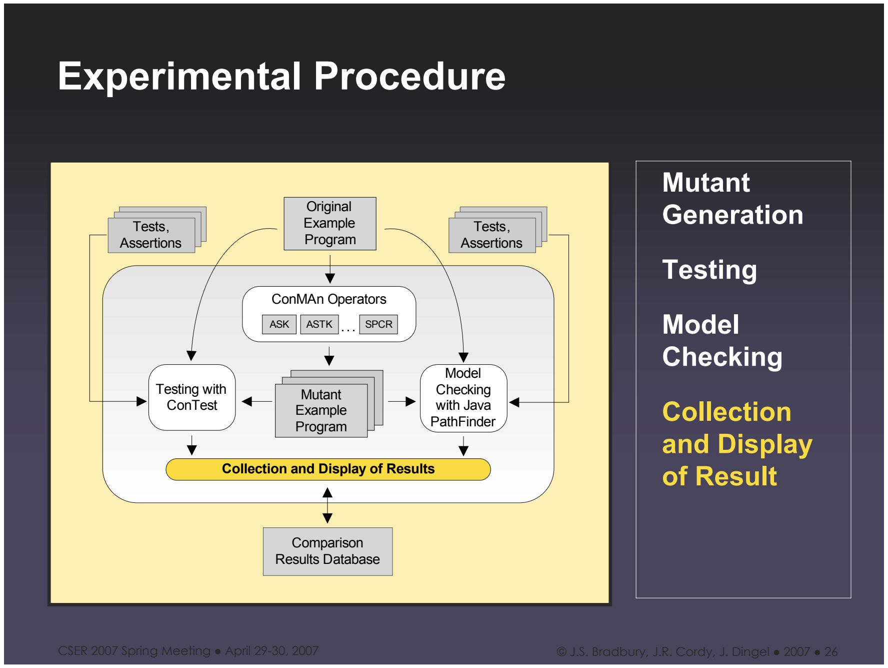


Experimental Procedure

Mutant Generation

Testing

Model Checking



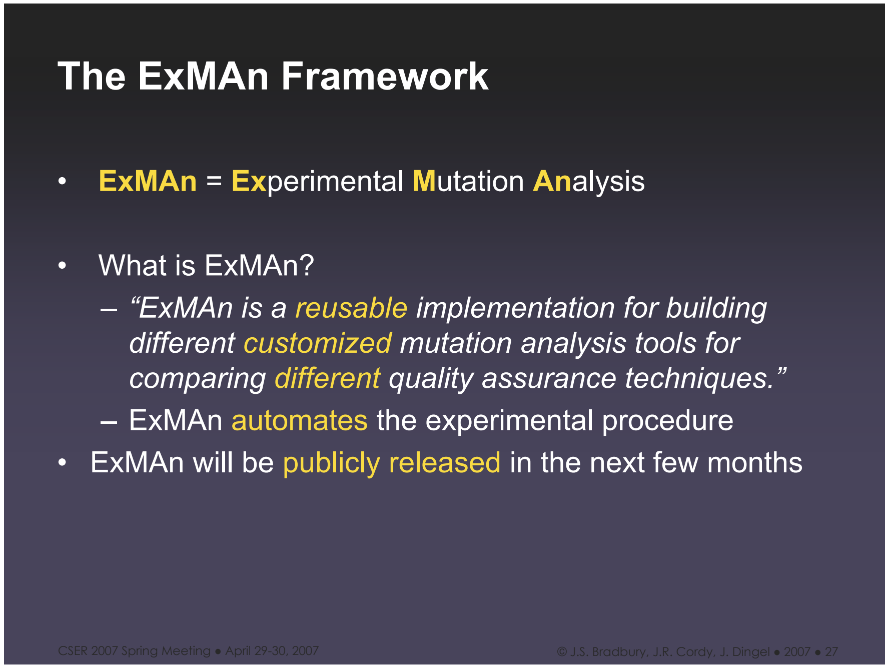
Experimental Procedure

Mutant Generation

Testing

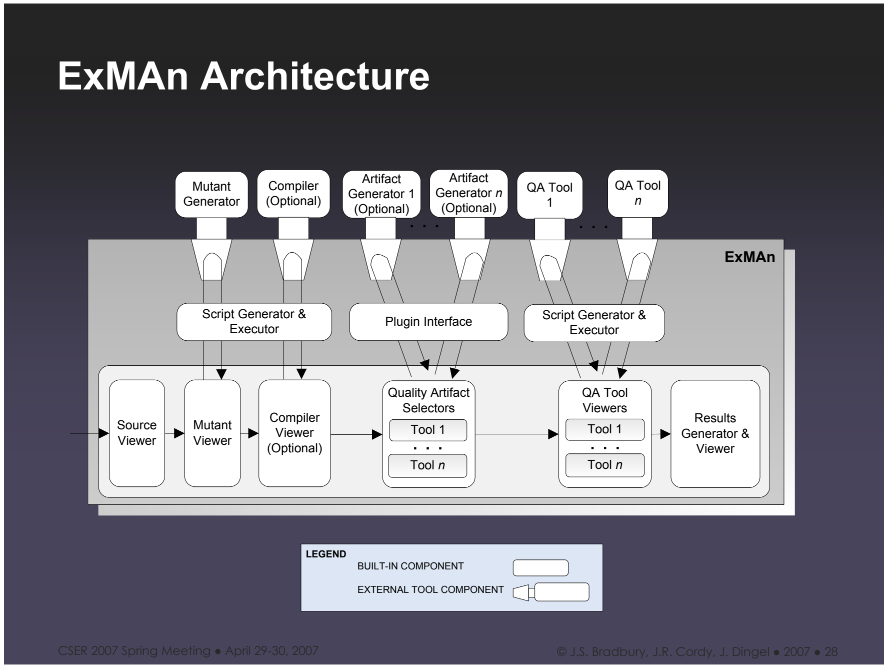
Model Checking

Collection and Display of Result



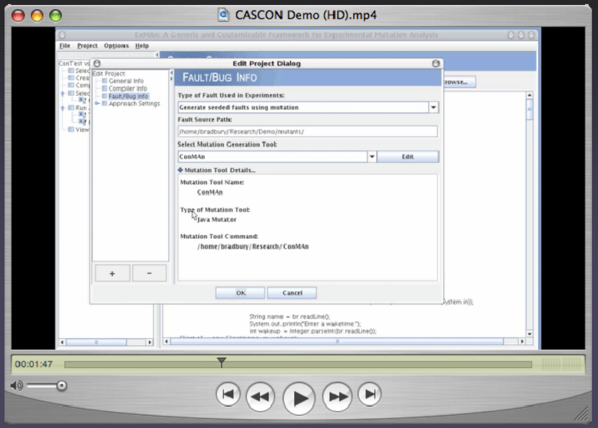
The ExMAn Framework

- ExMAn = Experimental Mutation Analysis
- What is ExMAn?
  - *“ExMAn is a reusable implementation for building different customized mutation analysis tools for comparing different quality assurance techniques.”*
  - ExMAn automates the experimental procedure
- ExMAn will be publicly released in the next few months



ExMAn Architecture

Video Demo



FAULT/BUG INFO

Type of Fault Used in Experiments: Generate seeded faults using mutation

Fault Source Path: /home/bradbury/Research/Demo/mutants/

Select Mutation Generation Tool: ConMAn

Mutation Tool Details...

Mutation Tool Name: ConMAn

Type of Mutation Tool: Java Mutator

Mutation Tool Command: /home/bradbury/Research/ConMAn

```
String name = br.readLine();
System.out.println("Enter a waketime:");
int wakeup = Integer.parseInt(br.readLine());
```

- Available at:
  http://www.cs.queensu.ca/~bradbury/videos/CASCON2006.mp4

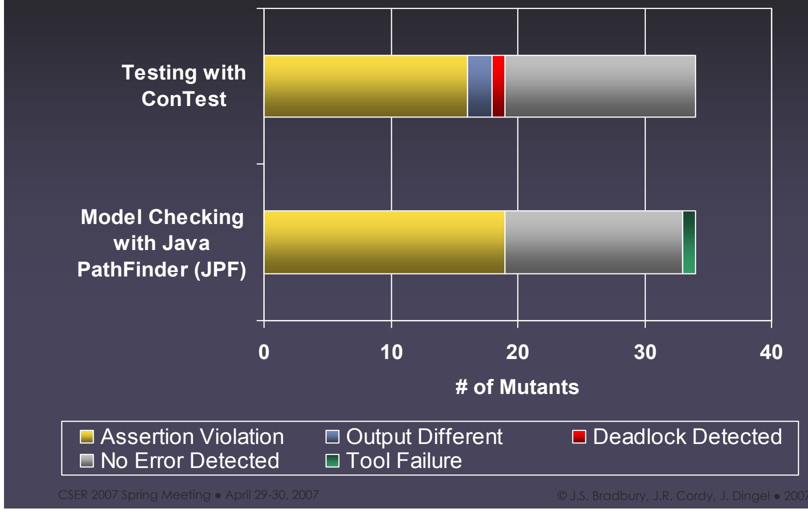
ConTest vs. Java PathFinder

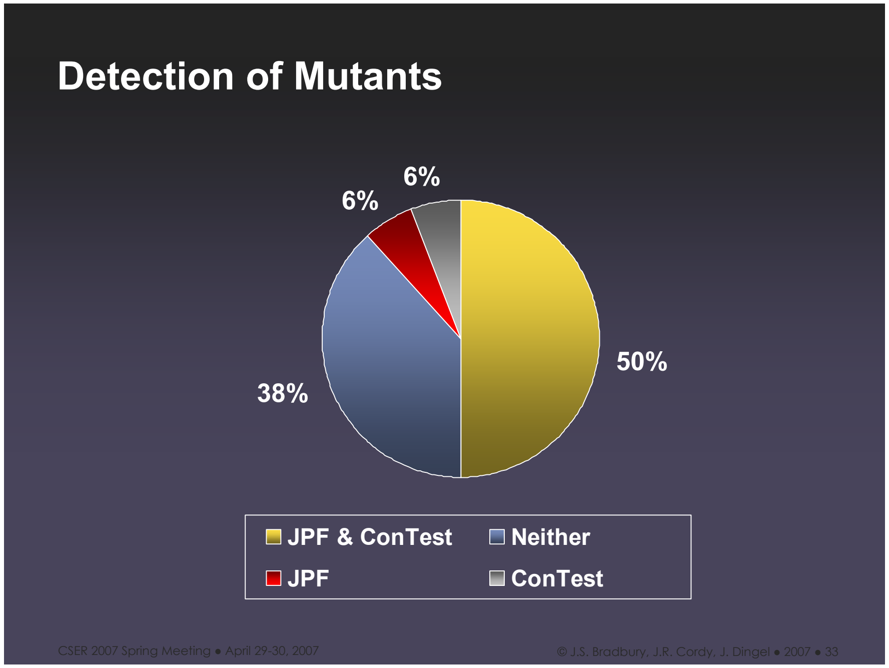
- How do we better understand the **effectiveness** of each technique?
  - We measure the mutant score for each technique (dependent variable)
  - We vary the analysis technique (factor)
  - We fix all other independent variables
    - quality artifacts (tests and properties), example programs …

Example Programs

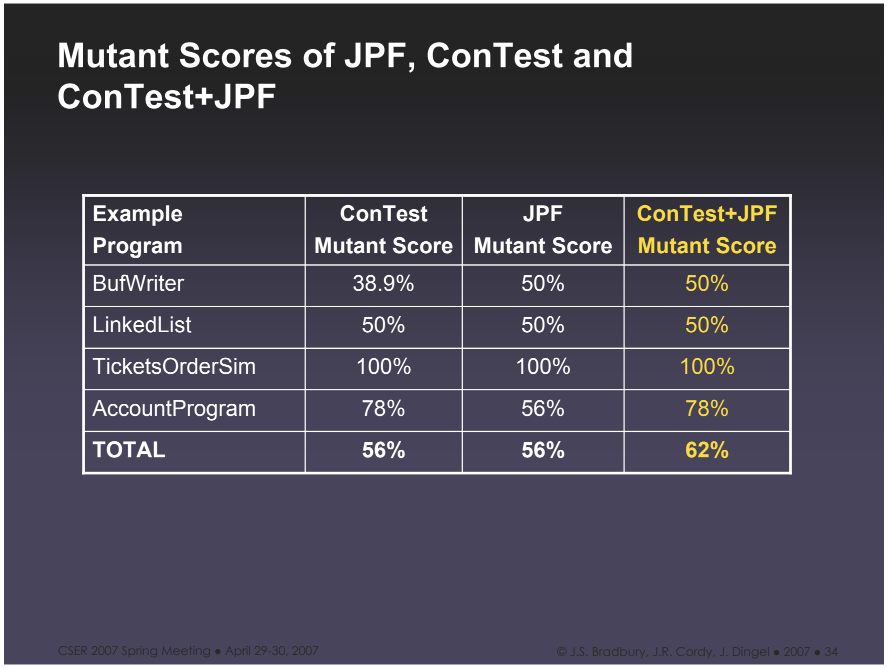
- Ticket Order Simulation
  - Simulates multiple agents selling tickets for a flight
- Linked List
  - Involves storing data in a concurrent linked list (data structure)
- Buffered Writer
  - Two different types of writer threads are updated a buffer that is being read by a reader thread
- Account Management System
  - Manages a series of transactions between a number of accounts

Quantity of Mutants Killed



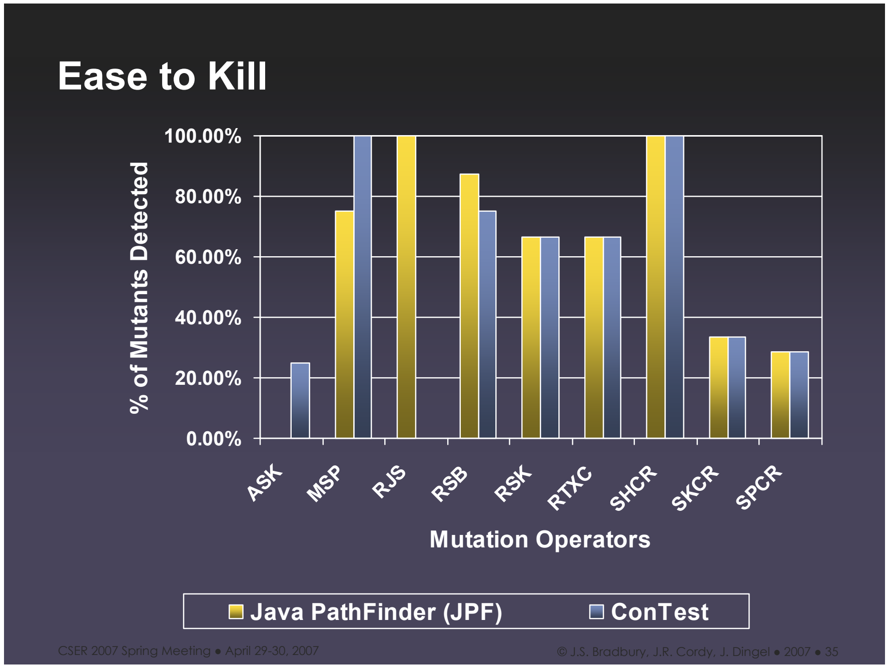


Detection of Mutants

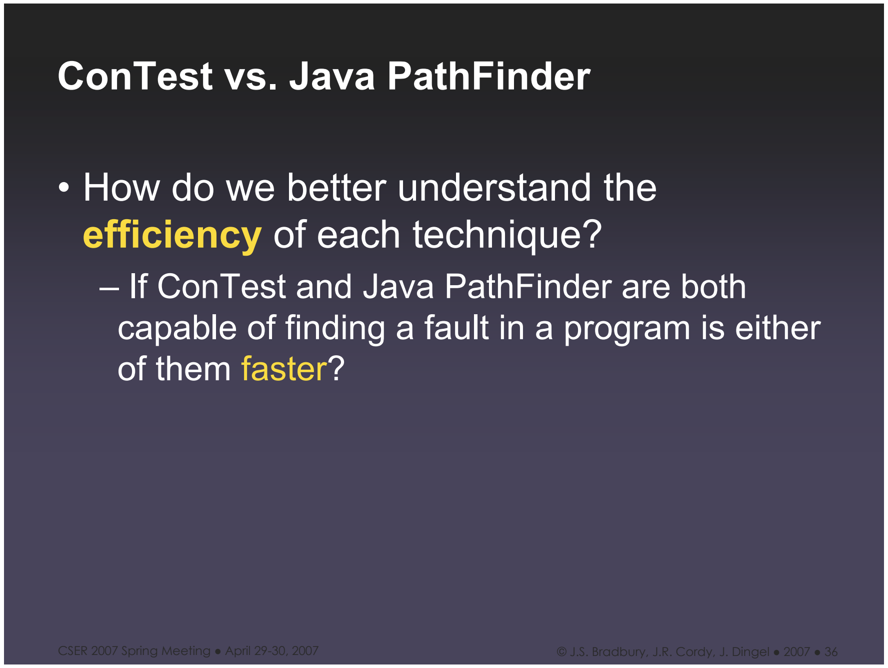


Mutant Scores of JPF, ConTest and ConTest+JPF

Example Program	ConTest Mutant Score	JPF Mutant Score	ConTest+JPF Mutant Score
BufWriter	38.9%	50%	50%
LinkedList	50%	50%	50%
TicketsOrderSim	100%	100%	100%
AccountProgram	78%	56%	78%
TOTAL	**56%**	**56%**	**62%**



Ease to Kill



ConTest vs. Java PathFinder

- How do we better understand the **efficiency** of each technique?
  - If ConTest and Java PathFinder are both capable of finding a fault in a program is either of them faster?

ConTest vs. Java PathFinder

- **Experimental Setup**
  - –*null hypothesis (H_0):* Time to detect a fault for JPF > Time to detect a fault for ConTest
  - –*dependent variable(s):* analysis time
  - –*independent variables:*
    - *factor:* analysis technique
    - *fixed:* quality artifacts (tests and properties) software under evaluation

ConTest vs. Java PathFinder

- Time for ConTest (seconds)
  - Mean = 2.0314
  - Median = 1.2030
- Time for Java PathFinder (seconds)
  - Mean = 3.2835
  - Median = 2.3320
- Conducted a paired t-test for n=19
  - P-value = 0.0085 (reject H_O at the 0.05 level)
  - JPF is not more efficient than ConTest for our example programs

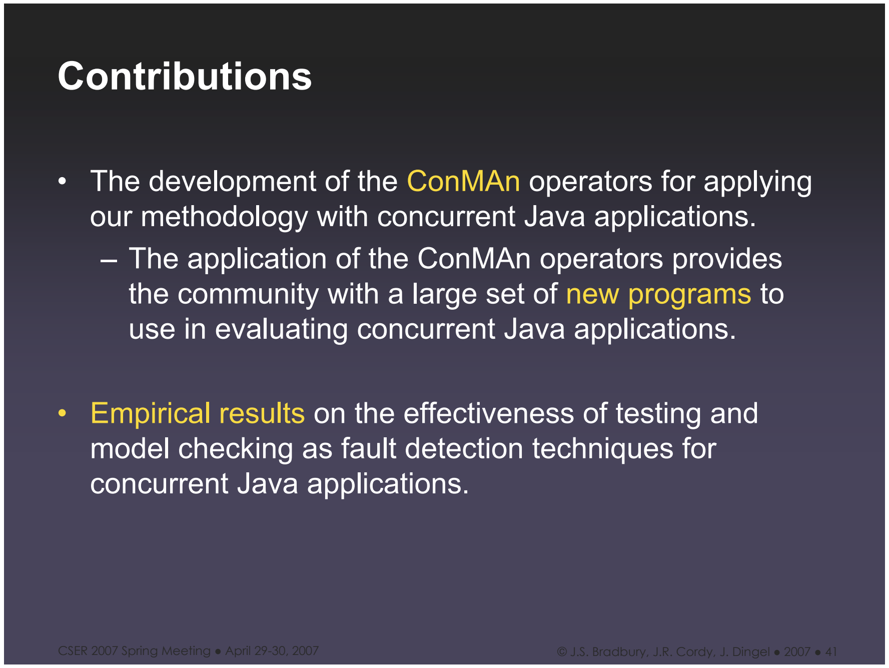
Threats to Validity

- internal validity
- external validity:
  - –Threats to external validity include:
    - the software being experimented on is not representative of software in general
    - the mutant faults do not adequately represent real faults for the programs under experiment
- construct validity
- conclusion validity

Contributions

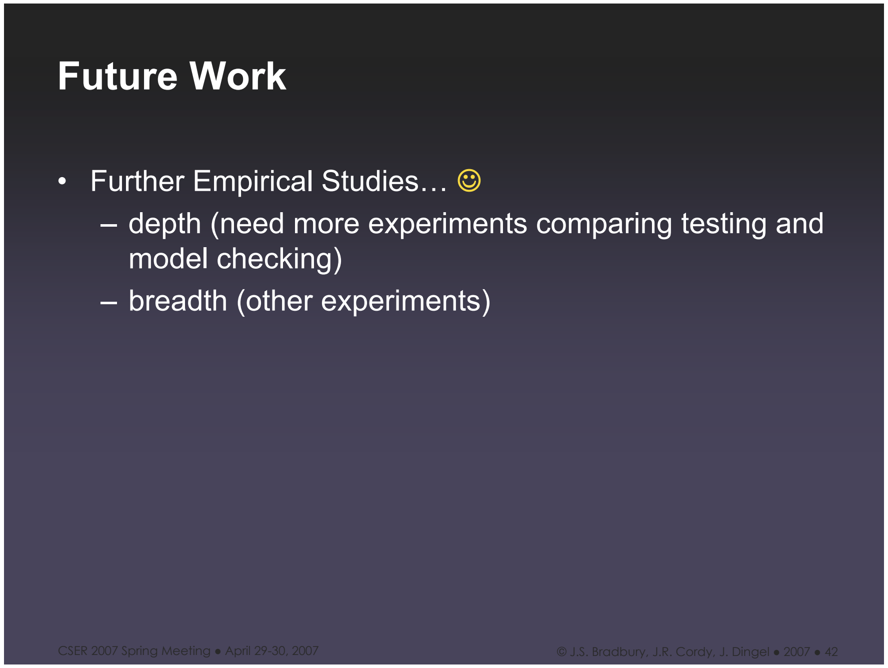
- A set of generalized mutation-based methods for conducting controlled experiments of different quality assurance approaches with respect to fault detection.

- The implementation of the ExMAn framework to automate and support our methodology.
  - The contribution of ExMAn includes its abilities to act as an enabler for further research



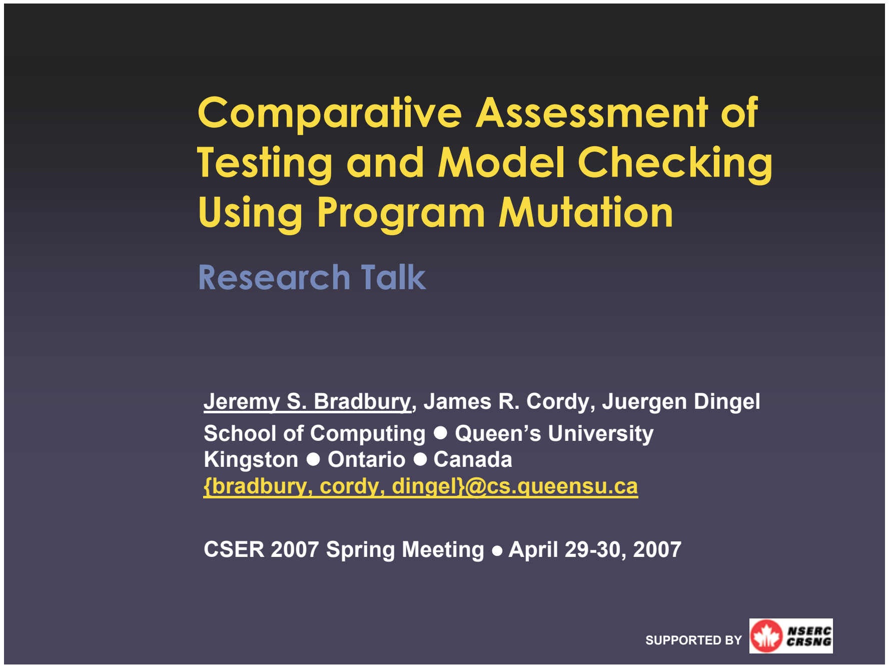
Contributions

- The development of the ConMAn operators for applying our methodology with concurrent Java applications.
  - The application of the ConMAn operators provides the community with a large set of new programs to use in evaluating concurrent Java applications.
- Empirical results on the effectiveness of testing and model checking as fault detection techniques for concurrent Java applications.



Future Work

- Further Empirical Studies… ☺
  - depth (need more experiments comparing testing and model checking)
  - breadth (other experiments)



Comparative Assessment of Testing and Model Checking Using Program Mutation

Research Talk

Jeremy S. Bradbury, James R. Cordy, Juergen Dingel
School of Computing ● Queen's University
Kingston ● Ontario ● Canada
{bradbury, cordy, dingel}@cs.queensu.ca

CSER 2007 Spring Meeting ● April 29-30, 2007

SUPPORTED BY NSERC CRSNG