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

> ...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 [S105]

How can we ensure concurrent programs are bug free?
Conducting 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)
Experimental Setup

The ConMAn Operators

- **ConMAn = Concurrency Mutation Analysis**
- 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

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>
...
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>
    ...
}
```

No Lock Bug!

---

Example ConMAn Mutation
ESP – Exchange Synchronized Block Parameters

```
Object lock1 = new Object();
Object lock2 = new Object();
...
synchronized (lock1) {
    synchronized (lock2) {
        <statement c1>
        <statement c2>
    }
    <statement n1>
    ...

Object lock1 = new Object();
Object lock2 = new Object();
...
synchronized (lock1) {
    synchronized (lock2) {
        <statement c1>
        <statement c2>
    }
    <statement n2>
    ...
```

Deadlock bug!
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
Video Demo

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

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 ...

Quantity of Mutants Killed
Detection of Mutants

- 6% of Mutants Detected
- Java PathFinder (JPF)
- ConTest
- Neither
- JPF & ConTest

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?

Mutant Scores of JPF, ConTest and ConTest+JPF

<table>
<thead>
<tr>
<th>Example Program</th>
<th>ConTest Mutant Score</th>
<th>JPF Mutant Score</th>
<th>ConTest+JPF Mutant Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>BufWriter</td>
<td>38.9%</td>
<td>50%</td>
<td>50%</td>
</tr>
<tr>
<td>LinkedList</td>
<td>50%</td>
<td>50%</td>
<td>50%</td>
</tr>
<tr>
<td>TicketsOrderSim</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>AccountProgram</td>
<td>78%</td>
<td>56%</td>
<td>78%</td>
</tr>
<tr>
<td>TOTAL</td>
<td>56%</td>
<td>56%</td>
<td>62%</td>
</tr>
</tbody>
</table>
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 $H_0$ at the 0.05 level)
  - JPF is not more efficient than ConTest for our example programs

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

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

- 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

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