Research in Concurrent Software Testing:
A Systematic Review

Simone Souza\textsuperscript{1}
Maria Brito\textsuperscript{1}  Rodolfo Silva\textsuperscript{1}  Paulo Souza\textsuperscript{1}  Ed Zaluska\textsuperscript{2}

\textsuperscript{1}Universidade de S\~{a}o Paulo - Brazil
\textsuperscript{2}University of Southampton - UK
srocio@icmc.usp.br

PADTAD 2011, Toronto, Canada, July 17, 2011
Outline

1. Systematic Review
2. Planning and Conduction
3. Results
4. Conclusion
Systematic Review

- What is a systematic review?
- Why do I need of a systematic review?
- How to do a systematic review?
A systematic literature review is a means of identifying, evaluating and interpreting all available research relevant to a particular research question, or topic area, or phenomenon of interest [Kitchenham and Charters, 2007].
Reasons for performing a Systematic Literature Review

- Most research starts with a literature review: this review must be as complete as possible!
- To summarise the available existing evidence for a research area
  - Evidences about: most used programming language, testing tool...
- To identify any gaps in current research and suggest areas for further investigation
This process is based on systematic review used in medicine area

Kitchenham has adapted this process for Software Engineering area [Kitchenham, 2004, Kitchenham and Charters, 2007]
Systematic Review process

1. **Planning the review**
   - Identification of the need for a review
   - Specifying the research question(s)
   - Definition of inclusion and exclusion criteria
   - Developing a review protocol

2. **Conducting the review**
   - Selection of the primary studies
   - Data extraction and synthesis

3. **Reporting the review**
1. Definition of the **research questions** and **search strings**
   - The most important part of any systematic review
   - The research questions drive the entire systematic review methodology
   - The review results are strictly related to the quality of the search strings
Systematic review planning

1. Research questions defined in our review protocol:

- *What testing approaches have been proposed to test concurrent programs?*
- *What fault taxonomies related to concurrent programs have been identified?*
- *What tools have been developed to test concurrent programs?*
2. Search strings defined in our review protocol:

- String 1: ("parallel program*" OR "concurrent program*" OR "multithread*" OR "multi-thread*") AND ("test*")
- String 2: ("concurren*" OR "parallel") AND ("bug*" OR "defect*" OR "error*") AND ("taxonomy")
3. Selection of the digital source databases:

- ACM Digital Library (portal.acm.org)
- IEEE eXplore (ieeexplore.ieee.org)
- SCOPUS (scopus.com)
- CITESEER (citeseerx.ist.psu.edu)
4. Definition of a **Control List**

- Composed of already-known primary studies related to the research subject
- Used to check if the research results contain all papers present in this control list
- Provides a mechanism to evaluate the effectiveness of the search strings
5. Definition of selection criteria

1. Inclusion criteria
   - (IC1) Primary studies that present testing approaches for concurrent programs
   - (IC2) Primary studies that characterize specific bugs related to concurrent programs
   - (IC3) Primary studies that propose tools for supporting concurrent program testing

2. Exclusion criteria
   - (EC1) primary studies that present testing approaches not related to concurrent programs
   - (EC2) primary studies that present fault types not related to concurrent programs
   - (EC3) primary studies that propose testing tools not related to concurrent program program testing
The review was performed between April/2011 and May/2011

Searches returned **1166 primary studies**

JabRef\(^1\) was used during the selection process

- JabRef tool is an open source bibliography reference manager.
- The native file format used is BibTeX, the standard LaTeX bibliography format.

\(^1\)http://jabref.sourceforge.net/index.php
Systematic review execution

- Four participants performed the primary studies selection
- Two selection iterations were performed:
  - Initial selection: 314 primary studies selected
  - Final selection: 175 primary studies selected
Systematic review execution

- Classification of the primary studies:
  - Testing approach (136)
  - Testing tool (43)
  - Fault taxonomy (7)
  - Others - surveys, benchmarks, experimental studies (5)
Result: Testing approach for multithreaded programs

<table>
<thead>
<tr>
<th>Technique</th>
<th>Static/dynamic</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>mutation testing, failure injection</td>
<td>static</td>
<td>[Artho et al., 2006, Gilgoric et al., 2010, Bradbury et al., 2006, Sen and Abadir, 2010]</td>
</tr>
<tr>
<td>model-based testing</td>
<td>static/dynamic</td>
<td>[Aichernig et al., 2009a, Aichernig et al., 2009b, Campbell et al., 2005, Chen, 2000b, Seo et al., 2006, Sohn et al., 1999]</td>
</tr>
<tr>
<td>static analysis, symbolic analysis</td>
<td>static/dynamic</td>
<td>[Kundu et al., 2010, Rungta et al., 2009, Chen et al., 2009, Chen, 2009, Chen et al., 2000, Flanagan et al., 2005]</td>
</tr>
<tr>
<td>execution analysis</td>
<td>dynamic</td>
<td>[Krena et al., 2010, Barnhart et al., 2008, McMinn, 2009, Burckhardt et al., 2010, Sen and Agha, 2006]</td>
</tr>
<tr>
<td>testing driven development</td>
<td>static/dynamic</td>
<td>[Dantas et al., 2008a, Ricken, 2007, Ricken and Cartwright, 2009, Jagannath et al., 2010a]</td>
</tr>
</tbody>
</table>
### Result: Testing approach for message-passing programs

<table>
<thead>
<tr>
<th>Technique</th>
<th>Static/dynamic</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Koppol and Tai, 1996, Koppol et al., 2002, Souza et al., 2008b,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Kojima et al., 2009, Krawczyk and Wiszniewski, 1996, Wang et al., 1997,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Shih et al., 1996, Katayama et al., 1999, Katayama et al., 1997,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Katayama et al., 1995, Katayama et al., 1996, Katayama et al., 1998,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Liang et al., 2000, Yuan et al., 2006]</td>
</tr>
<tr>
<td>mutation testing</td>
<td>static</td>
<td>[Jagannath et al., 2010b]</td>
</tr>
<tr>
<td>model-based testing</td>
<td>static</td>
<td>[Chung et al., 1999]</td>
</tr>
<tr>
<td>controlled execution</td>
<td>dynamic</td>
<td>[Oberhuber and Munchen, 1995, Vuduc et al., 2006, Tai and Karacali, 2001,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Lei, 2003, Olsson, 1999]</td>
</tr>
<tr>
<td>race detection</td>
<td>dynamic</td>
<td>[Jianxin and Dingxing, 1996]</td>
</tr>
<tr>
<td>static analysis</td>
<td>static</td>
<td>[Christakis and Sagonas, 2011]</td>
</tr>
</tbody>
</table>
**Result: Testing approach for both paradigms**

<table>
<thead>
<tr>
<th>Technique</th>
<th>Static/dynamic</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>test case generation</td>
<td>static/dynamic</td>
<td>[Ding et al., 2008, Tan et al., 2009, Xiaoan et al., 2009]</td>
</tr>
</tbody>
</table>
## Result: Concurrent software testing tool

<table>
<thead>
<tr>
<th>Language</th>
<th>Tool</th>
<th>Technique</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ada</td>
<td>CATS</td>
<td>reachability analysis</td>
<td>[Young et al., 1992]</td>
</tr>
<tr>
<td></td>
<td>TSG</td>
<td>formal verification</td>
<td>[Carver and Durham, 1995]</td>
</tr>
<tr>
<td></td>
<td>TCgen</td>
<td>structural testing</td>
<td>[Katayama et al., 1998]</td>
</tr>
<tr>
<td>MPI</td>
<td>GEM</td>
<td>dynamic verification of relevant interleavings</td>
<td>[Humphrey et al., 2010]</td>
</tr>
<tr>
<td></td>
<td>JitterBug</td>
<td>controlled execution</td>
<td>[Vuduc et al., 2006]</td>
</tr>
<tr>
<td></td>
<td>MPIRace-Check</td>
<td>race detection</td>
<td>[Park et al., 2007]</td>
</tr>
<tr>
<td></td>
<td>MARMOT</td>
<td>race and deadlock detection</td>
<td>[Krammer et al., 2004]</td>
</tr>
<tr>
<td>PVM/MPI</td>
<td>ValiPar</td>
<td>structural testing</td>
<td>[Souza et al., 2008b]</td>
</tr>
<tr>
<td>PVM</td>
<td>ValiPVM</td>
<td>structural testing</td>
<td>[Souza et al., 2008a]</td>
</tr>
<tr>
<td></td>
<td>STEP</td>
<td>structural testing</td>
<td>[Krawczyk and Wiszniewski, 1996]</td>
</tr>
<tr>
<td>C</td>
<td>monitoring tool</td>
<td>Ordered Sequence Testing Criterion</td>
<td>[Itoh et al., 1996]</td>
</tr>
<tr>
<td></td>
<td>Inspect</td>
<td>model checker</td>
<td>[Yang et al., 2008]</td>
</tr>
<tr>
<td></td>
<td>C2Petri</td>
<td>formal verification</td>
<td>[Kavi et al., 2002]</td>
</tr>
<tr>
<td></td>
<td>Direct</td>
<td>sensitivity analysis and runtime manager</td>
<td>[Chatterjee et al., 2010]</td>
</tr>
<tr>
<td>C/C++</td>
<td>ViP</td>
<td>formal verification</td>
<td>[Dingel and Liang, 2004]</td>
</tr>
<tr>
<td></td>
<td>ConMem</td>
<td>race and atomicity detection</td>
<td>[Zhang et al., 2010]</td>
</tr>
<tr>
<td>C++</td>
<td>MultiRace</td>
<td>race detection</td>
<td>[Pozniansky and Schuster, 2003]</td>
</tr>
<tr>
<td></td>
<td>ThreadSanitizer</td>
<td>race detection</td>
<td>[Serebrany and Iskhodzhanov, 2009]</td>
</tr>
<tr>
<td>.NET framework</td>
<td>GAMBIT</td>
<td>formal verification</td>
<td>[Coons et al., 2010]</td>
</tr>
<tr>
<td></td>
<td>CHESS</td>
<td>systematic testing</td>
<td>[Ball et al., 2009b, Musuvathi et al., 2007, Musuvathi et al., 2008, Musuvathi and Qadeer, 2007, Ball et al., 2009a, Musuvathi and Qadeer, 2008]</td>
</tr>
</tbody>
</table>
### Result: Concurrent software testing tool II

<table>
<thead>
<tr>
<th>Language</th>
<th>Tool</th>
<th>Technique</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java</td>
<td>jCUTE</td>
<td>race detection</td>
<td>[Sen and Agha, 2006]</td>
</tr>
<tr>
<td></td>
<td>RichTest</td>
<td>reachability testing</td>
<td>[Lei and Carver, 2006, Carver and Lei, 2004]</td>
</tr>
<tr>
<td></td>
<td>MuTMuT</td>
<td>mutation testing</td>
<td>[Gligoric et al., 2010]</td>
</tr>
<tr>
<td></td>
<td>ConTest</td>
<td>controlled execution</td>
<td>[Edelstein et al., 2003, Farchi et al., 2003, Copty and Ur, 2005, Krena et al., 2010, Krena et al., 2007]</td>
</tr>
<tr>
<td></td>
<td>ThreadControl</td>
<td>testing driven development</td>
<td>[Dantas et al., 2008a]</td>
</tr>
<tr>
<td></td>
<td>Kivati</td>
<td>atomicity violation</td>
<td>[Chew and Lie, 2010]</td>
</tr>
<tr>
<td></td>
<td>HAVE</td>
<td>atomicity violation</td>
<td>[Chen et al., 2009]</td>
</tr>
<tr>
<td></td>
<td>CTrigger</td>
<td>atomicity violation</td>
<td>[Park et al., 2009]</td>
</tr>
<tr>
<td></td>
<td>PENEOPE</td>
<td>atomicity violation</td>
<td>[Sorrentino et al., 2010]</td>
</tr>
<tr>
<td></td>
<td>DEJAVU</td>
<td>replay testing</td>
<td>[deok Choi and Zeller, 2002]</td>
</tr>
<tr>
<td></td>
<td>ConCrash</td>
<td>replay testing and unit testing</td>
<td>[Luo et al., 2010]</td>
</tr>
<tr>
<td></td>
<td>Enforcer</td>
<td>failure injection</td>
<td>[Artho et al., 2006]</td>
</tr>
<tr>
<td></td>
<td>Java PathFinder, TIE</td>
<td>model checking</td>
<td>[Maheswara et al., 2010, Havelund and Rosu, 2001]</td>
</tr>
<tr>
<td></td>
<td>CalFuzzer</td>
<td>active testing</td>
<td>[Joshi et al., 2009, Sen, 2007]</td>
</tr>
<tr>
<td></td>
<td>Bandera</td>
<td>formal verification</td>
<td>[Corbett et al., 2000]</td>
</tr>
<tr>
<td></td>
<td>FindLocks</td>
<td>formal verification</td>
<td>[Rose et al., 2005]</td>
</tr>
<tr>
<td>POSIX Threads API</td>
<td>Kendo</td>
<td>deterministic testing</td>
<td>[Olszewski et al., 2009]</td>
</tr>
<tr>
<td></td>
<td>ValiPthread</td>
<td>structural testing</td>
<td>[Sarmanho et al., 2008]</td>
</tr>
<tr>
<td></td>
<td>Gadara</td>
<td>deadlock detection</td>
<td>[Wang et al., 2008]</td>
</tr>
<tr>
<td></td>
<td>Della Pasta</td>
<td>all-du-path testing</td>
<td>[Yang et al., 1998]</td>
</tr>
</tbody>
</table>
## Result: Concurrent software bugs

<table>
<thead>
<tr>
<th>Bugs</th>
<th>Paradigm</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>processor activation errors, processor coordination errors</td>
<td>multithreaded</td>
<td>[Sung, 1988]</td>
</tr>
<tr>
<td>errors and computation errors</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ordering bug, atomicity violation and deadlock</td>
<td>multithreaded</td>
<td>[Burckhardt et al., 2010]</td>
</tr>
<tr>
<td>interleaving error and assumed never to occur, deadlock</td>
<td>multithreaded</td>
<td>[Farchi et al., 2003]</td>
</tr>
<tr>
<td>code segment unprotected</td>
<td></td>
<td></td>
</tr>
<tr>
<td>atomicity-violation or order-violation, others, bug manifestation</td>
<td>multithreaded</td>
<td>[Lu et al., 2008]</td>
</tr>
<tr>
<td>concerns to transacional memory and deadlock</td>
<td></td>
<td></td>
</tr>
<tr>
<td>interleaving error, data race error, wrong def-use relation</td>
<td>multithreaded</td>
<td>[Lu et al., 2007]</td>
</tr>
<tr>
<td>faults in monitors: enter, wait, signal-exit and internal process</td>
<td>multithreaded</td>
<td>[Cao et al., 2001]</td>
</tr>
<tr>
<td>termination faults; inconsistent states; logic and runtime faults</td>
<td></td>
<td></td>
</tr>
<tr>
<td>observability and locking error</td>
<td>both</td>
<td>[Bogdan et al., 1994]</td>
</tr>
<tr>
<td>testing patterns for software transactional memory (considering</td>
<td>multithreaded</td>
<td>[Lourenço and Cunha, 2007]</td>
</tr>
<tr>
<td>transactions and variables)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Contributions by research subject

Quantity the papers by research subject

- Testing approach: 68%
- Testing tool: 26%
- Bug taxonomy: 4%
- Others: 2%
Results

Quantity of Testing Tools by year (1992 - 2011)
What programming languages are supported by testing tools?

- Java: 36%
- MPI/PVM: 16%
- C/C++: 18%
- .NET: 5%
- Erlang: 5%
- Pascal: 2%
- Posix threads: 9%
- Ada: 7%
- CML: 2%
Results

Quantity of Testing Approaches by year (1989 - 2011)
Results

Quantity of papers about testing approach: message-passing x shared memory paradigms

message-passing  multithreaded
Geography distribution of the authors

- USA: 89
- Canada: 13
- China: 16
- Japan: 11
- Taiwan: 6
- Austria: 2
- Brazil: 6
- Czech Republic: 2
- Denmark: 1
- France: 2
- Germany: 2
- Greece: 1
- Hong Kong: 1
- Israel: 7
- Italy: 1
- Korea: 3
- Lebanon: 1
- Mexico: 1
- Poland: 2
- Portugal: 1
- Russia: 1
- South Korea: 1
- Switzerland: 1
- Turkey: 1
- United Arab Emirates: 1
- UK: 1
Authors Relationship Diagram
Distribution of the primary studies by publication channel

<table>
<thead>
<tr>
<th>Publication channel</th>
<th>Publisher</th>
<th>Type</th>
<th>Number</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Asia Pacific Software Engineering Conference</td>
<td>IEEE</td>
<td>conference</td>
<td>9</td>
<td>5,56</td>
</tr>
<tr>
<td><strong>International Workshop on Parallel and Distributed Systems: Testing and Debugging (PADTAD)</strong></td>
<td>ACM</td>
<td>conference</td>
<td>8</td>
<td>4,94</td>
</tr>
<tr>
<td>International Conference on Architectural support for programming languages and operating systems</td>
<td>ACM</td>
<td>conference</td>
<td>7</td>
<td>4,32</td>
</tr>
<tr>
<td>Concurrency and Computation: Practice and Experience</td>
<td>Elsevier</td>
<td>journal</td>
<td>5</td>
<td>3,09</td>
</tr>
<tr>
<td>IEEE Transactions on Software Engineering</td>
<td>IEEE</td>
<td>journal</td>
<td>5</td>
<td>3,09</td>
</tr>
<tr>
<td>International Conference on Software Engineering</td>
<td>ACM</td>
<td>conference</td>
<td>5</td>
<td>3,09</td>
</tr>
<tr>
<td>ACM SIGSOFT Symposium on the Foundations of Software Engineering</td>
<td>ACM</td>
<td>conference</td>
<td>4</td>
<td>2,47</td>
</tr>
<tr>
<td>International Conference on Software Testing, Verification and Validation</td>
<td>IEEE</td>
<td>conference</td>
<td>4</td>
<td>2,47</td>
</tr>
<tr>
<td>International Symposium on Software Engineering for Parallel and Distributed Systems (PDSE)</td>
<td>IEEE</td>
<td>conference</td>
<td>4</td>
<td>2,47</td>
</tr>
<tr>
<td>Electronic Notes in Theoretical Computer Science</td>
<td>Elsevier</td>
<td>journal</td>
<td>3</td>
<td>1,85</td>
</tr>
<tr>
<td>International Computer Software and Applications Conference</td>
<td>IEEE</td>
<td>conference</td>
<td>3</td>
<td>1,85</td>
</tr>
<tr>
<td>International Conference on Automated Software Engineering</td>
<td>IEEE</td>
<td>conference</td>
<td>3</td>
<td>1,85</td>
</tr>
<tr>
<td>International Conference on Computer Aided Verification CAV</td>
<td>LNCS</td>
<td>conference</td>
<td>3</td>
<td>1,85</td>
</tr>
<tr>
<td>International Conference Fundamental Approaches to Software Engineering (FASE)</td>
<td>LNCS</td>
<td>conference</td>
<td>3</td>
<td>1,85</td>
</tr>
<tr>
<td>International Haifa Verification Conference</td>
<td>LNCS</td>
<td>conference</td>
<td>3</td>
<td>1,85</td>
</tr>
<tr>
<td>International Parallel and Distributed Processing Symposium</td>
<td>IEEE</td>
<td>conference</td>
<td>3</td>
<td>1,85</td>
</tr>
<tr>
<td>ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming</td>
<td>ACM</td>
<td>conference</td>
<td>2</td>
<td>1,23</td>
</tr>
<tr>
<td>ACM SIGSOFT International Symposium on Software Testing and Analysis</td>
<td>ACM</td>
<td>conference</td>
<td>2</td>
<td>1,23</td>
</tr>
<tr>
<td>Asia-Pacific Conference on Quality Software</td>
<td>IEEE</td>
<td>conference</td>
<td>2</td>
<td>1,23</td>
</tr>
<tr>
<td>IEEE International Conference on Systems, Man and Cybernetics</td>
<td>IEEE</td>
<td>conference</td>
<td>2</td>
<td>1,23</td>
</tr>
<tr>
<td>Information and Software Technology</td>
<td>Elsevier</td>
<td>journal</td>
<td>2</td>
<td>1,23</td>
</tr>
<tr>
<td>International Conference on Computational Science</td>
<td>LNCS</td>
<td>conference</td>
<td>2</td>
<td>1,23</td>
</tr>
<tr>
<td>International Conference on Dependable Systems and Networks</td>
<td>IEEE</td>
<td>conference</td>
<td>2</td>
<td>1,23</td>
</tr>
<tr>
<td>International Conference on Formal Engineering Methods</td>
<td>LNCS</td>
<td>conference</td>
<td>2</td>
<td>1,23</td>
</tr>
</tbody>
</table>
Difficulties and Limitations

- Definition of the search string:
  - the words of the strings must represent the expected primary studies set (hard task!)
  - the search string must be adapted for each source database
- The abstracts, titles and key words do not always represent the subject of the paper
  - Example: debugging X testing
- The selection of the primary studies is done manually
Conclusions

Suggestions for further investigation:

- Cost/efficacy evaluation of the proposed testing techniques;
- Definition of benchmarks to support testing technique evaluation;
- Integration of the testing approaches, considering different testing levels;
- Automatic generation of test data;
Conclusions

- The systematic review identifies different groups of authors working in important and challenging subjects:
  - Coverage testing, model checking, controlled execution, bug classification and testing tools
- 76% of the primary studies are contributions for the definition of new approaches for concurrent program testing
- The results confirm that the PADTAD Workshop has selected significant papers in current research areas
Thank you for your attention!
Simone Souza
Universidade de São Paulo
srocio@icmc.usp.br

PADTAD 2011, Toronto, Canada, July 17, 2011
Bibliography I


Classification of software defects in parallel programs.
Technical report, Faculty of Electronics, Technical University of Gdansk, Poland.

Dynamic analysis of java applications for multithreaded antipatterns.

Mutation operators for concurrent Java (J2SE 5.0).
*Workshop on Mutation Analysis*, page 11.

Applications of synchronization coverage.

A randomized scheduler with probabilistic guarantees of finding bugs.

Control of nondeterminism in testing distributed multithreaded programs.

Multiplexing of partially ordered events.
A robust monitor construct with runtime fault detection.  

Run-time fault detection in monitor based concurrent programming. 

Integrating formal methods and testing for concurrent programs.  

A stateful approach to testing monitors in multithreaded programs.  

A general model for reachability testing of concurrent programs.  

Distributed reachability testing of concurrent programs.  
Deterministic execution testing of concurrent ada programs.

Replay and testing for concurrent programs.

Analyzing the impact of change in multi-threaded programs.

Parametric and sliced causality.

A scheme for dynamic detection of concurrent execution of object-oriented software.

On verifying distributed multithreaded java programs.
*Hawaii International Conference on System Sciences (HICSS)*, page 8 pp. vol.1.

A study on static analysis in network of synchronizing fsms.
Bibliography V

Guided Testing of Concurrent Programs Using Value Schedules.
PhD thesis, University of Waterloo.

Testing concurrent programs using value schedules.
International Conference on Automated Software Engineering, pages 313–322.

Have: Detecting atomicity violations via integrated dynamic and static analysis.
International Conference Fundamental Approaches to Software Engineering (FASE), 5503:425–439.

An approach to analyzing dependency of concurrent programs.

Kivati: Fast detection and prevention of atomicity violations.

Detection of asynchronous message passing errors using static analysis.
International Symposium Practical Aspects of Declarative Languages (PADL), Austin, TX, USA, 6539 LNCS:5–18.

Task decomposition testing and metrics for concurrent programs.
International Symposium on Software Reliability Engineering, pages 122 –130.
Testing of concurrent programs based on message sequence charts.
International Symposium on Software Engineering for Parallel and Distributed Systems, pages 72–82.

Gambit: Effective unit testing for concurrency libraries.

Multi-threaded testing with AOP is easy, and it finds bugs!

Bandera: A source-level interface for model checking java programs.

Improving developers’ confidence in test results of multi-threaded systems: avoiding early and late assertions.

Improving automated testing of multi-threaded software.
Obtaining trustworthy test results in multi-threaded systems.
_Simpósio Brasileiro de Engenharia de Software_, pages 1–10.

Isolating failure-inducing thread schedules.

Dmp: deterministic shared memory multiprocessing.
_International Conference on Architectural support for programming languages and operating systems_, pages 85–96.

A rigorous approach towards test case generation.

Automating comprehensive safety analysis of concurrent programs using verisoft and txl.
_ACMSIGSOFT International Symposium on the Foundations of Software Engineering_.

Framework for testing multi-threaded java programs.

Delay-bounded scheduling.
_Symposium on Principles of Programming Languages_, pages 411–422.
Bibliography VIII

Explaining intermittent concurrent bugs by minimizing scheduling noise.
*International Haifa Verification Conference, HVC, 4383 LNCS:*183–197.

Effective testing and debugging techniques for a group communication system.
*International Conference on Dependable Systems and Networks,* pages 80–85.

Concurrent bug patterns and how to test them.
*International Parallel and Distributed Processing Symposium,* page 7 pp.

Distributed online software monitoring of manycore architectures.

Exploiting purity for atomicity.

Mutmut: Efficient exploration for mutation testing of multithreaded code.
*International Conference on Software Testing, Verification and Validation (ICST),* pages 55 –64.

On testing multi-threaded java programs.
Model checking multithreaded programs by means of reduced models using a tool based on the selective
mu-calculus logic to check systems described through the ccs specification language.

Java PathExplorer - a runtime verification tool.
*International Symposium on Artificial Intelligence, Robotics and Automation in Space: A New Space

GEM: graphical explorer of mpi programs.

Reachability testing: An approach to testing concurrent software.

A prototype of a concurrent behavior monitoring tool for testing of concurrent programs.

Imunit: improved multithreaded unit testing.
Mutation operators for actor systems.

Analyzing nondeterminacy of message passing programs.

Systematic testing of multithreaded programs.

Calfuzzer: An extensible active testing framework for concurrent programs.

Enforcing textual alignment of collectives using dynamic checks.

Automated test sequence generation using sequencing constraints for concurrent programs.

Event interactions graph for test-case generations of concurrent programs.
A method for structural testing of ada concurrent programs using the event interactions graph.  

Test-case generation method for concurrent programs including task-types.  

Design and implementation of test-case generation for concurrent programs.  

Test-case generation for concurrent programs with the testing criteria using interaction sequences.  

Modeling multithreaded applications using petri nets.  

Generalized symbolic execution for model checking and testing.  
Procedures for performing systematic reviews.
Technical Report TR/SE-0401, Keele University, UK.

Guidelines for performing systematic literature reviews in software engineering.

A model for concurrent states and its coverage criteria.

Incremental integration testing of concurrent programs.

An incremental approach to structural testing of concurrent software.

MPI application development using the analysis tool MARMOT.

A method for determining testing scenarios for parallel and distributed software.
Technical report, Faculty of Electronics, Telecommunications and Informatics, Technical University of Gdansk, Poland, Tech Report N. 193/1996.
Healing data races on-the-fly.

A platform for search-based testing of concurrent software.

Contessa: Concurrency testing augmented with symbolic analysis.

Non-deterministic testing of concurrent programs.

Reachability testing of semaphore-based programs.

Reachability testing of concurrent programs.
A combinatorial testing strategy for concurrent programs.

Li, J., Hei, D., and Yan, L. (2009).
Correctness analysis based on testing and checking for openmp programs.
ChinaGrid Annual Conference (ChinaGrid '09), pages 210 –215.

A framework of reachability testing for java multithread programs.

Timing-sequence testing of parallel programs.

Testing patterns for software transactional memory engines.

A study of interleaving coverage criteria.
Learning from mistakes: a comprehensive study on real world concurrency bug characteristics.

Avio: Detecting atomicity violations via access-interleaving invariants.
*International Conference on Architectural support for programming languages and operating systems (ASPLOS)*, pages 37–48.

A lightweight and portable approach to making concurrent failures reproducible.
*International Conference Fundamental Approaches to Software Engineering (FASE)*, 6013 LNCS:323–337.

Tie: An interactive visualization of thread interleavings.

Search-based failure discovery using testability transformations to generate pseudo-oracles.

Iterative context bounding for systematic testing of multithreaded programs.
*ACM SIGPLAN Conference on Programming language design and implementation*, 42:446–455.
Fair stateless model checking.
*ACM SIGPLAN Programming Language Design and Implementation (PLDI)*.

Chess: A systematic testing tool for concurrent software.

Finding and reproducing heisenbugs in concurrent programs.

Elimination of nondeterminacy for testing and debugging parallel programs.
*International Workshop on Automated and Algorithmic Debugging (AADEBUG’95)*.

Reproducible execution of sr programs.

Kendo: efficient deterministic multithreading in software.
*International Conference on Architectural support for Programming languages and operating systems, ACM SIGPLAN Notes*, 44:97–108.

MPIRace-check: Detection of message races in MPI programs.
Ctrigger: exposing atomicity violation bugs from their hiding places.

Efficient on-the-fly data race detection in multithreaded C++ programs.
*International Parallel and Distributed Processing Symposium*, page 8 pp.

A feasible strategy for reachability testing of internet-based concurrent programs.
*IEEE International Conference on Networking, Sensing and Control (ICNSC)*, pages 1559–1564.

Unit testing concurrent software.

Storm: static unit checking of concurrent programs.

ConcJUnit: Unit testing for concurrent programs.

A framework for testing concurrent programs.
Technical report, RICE UNIVERSITY.


State-based reproducible testing for CORBA applications.

Penelope: Weaving threads to expose atomicity violations.

ValiPVM - a graphical tool for structural testing of PVM programs.

Structural testing criteria for message-passing parallel programs.

Dynamic datarace detection for object-oriented programs.
Technical report, MASSACHUSETTS INSTITUTE OF TECHNOLOGY.

Testing concurrent java programs using randomized scheduling.

Testing shared-memory parallel programs.
*Symposium on the Frontiers of Massively Parallel Computation*, pages 559 –566.


Gadara: Dynamic deadlock avoidance for multithreaded programs.

Software testing and metrics for concurrent computation through task decomposition.

Testing path generation algorithm with network performance constraints for nondeterministic parallel programs.
International Conference on Web-Age Information Management Workshops, WAIM 2006.

Wong, W., Lei, Y., and Ma, X. (2005).
Effective generation of test sequences for structural testing of concurrent programs.

Reachability graph-based test sequence generation for concurrent programs.

Composable specifications for structured shared-memory communication.
International Conference on Object oriented Programming systems languages and applications, pages 140–159.
A self-adaptive test framework for concurrent programs.

Test case generation of concurrent programs based on event graph.
*International Joint Conference on INC, IMS, and IDC (NCM 2009)*, pages 143–149.

Formally defining a graphical language for monitoring and checking object interactions.
*International Conference Model Driven Engineering Languages and Systems, MoDELS*, 4735 LNCS:620–634.

Identifying redundant test cases for testing parallel language constructs.
*Technical Conference Constructs, ARL-ATIRP*.

An algorithm for all-du-path testing coverage of shared memory parallel programs.
*Asian Test Symposium*, pages 263–268.

All-uses testing of shared memory parallel programs.

All-du-path coverage for parallel programs.
The analysis of infeasible concurrent paths of concurrent ada programs.

A path analysis approach to concurrent program testing.

Path analysis testing of concurrent programs.

Inspect: A runtime model checker for multithreaded c programs.
Technical report, School of Computing University of Utah, n. UUCS-08-004.

Smt-based symbolic model checking for multi-threaded programs.
*Workshop Exploiting Concurrency Efficiently and Correctly*.

Trace-driven verification of multithreaded programs.

A concurrency analysis tool suite: Rationale, design, and preliminary experience.
Technical report, ACM Transactions on Software Engineering and Methodology.
A case for an interleaving constrained shared-memory multi-processor.

A graph-search based approach to BPEL4WS test generation.
*International Conference on Software Engineering Advances, ICSEA’06*.

Conmem: Detecting severe concurrency bugs through an effect-oriented approach.

Zhao, J. (1999).
Multithreaded dependence graphs for concurrent java programs.