Concurrent Software Testing: A Systematic Review*

Maria A. S. Brito, Katia R. Felizardo
Paulo S. L. Souza, and Simone R. S. Souza

Instituto de Ciências Matemáticas e de Computação
Universidade de São Paulo – (ICMC/USP)
Caixa Postal 668 – 13560-970 – São Carlos – SP – Brazil
{masbrit,katiarf,pssouza,srocio}@icmc.usp.br
http://www.icmc.usp.br

Abstract. Software testing applied for concurrent programs is a challenging activity. Considering the relevance of concurrent programs testing, several research have been conducted in this area, especially involving adaptation of the techniques and criteria applied in sequential programs. This technical report presents a systematic review, which is a technique coming from Evidence-Based Software Engineering, aiming to find out research about testing in this context. As result, the review identified testing criteria, bugs taxonomy and testing tools that can provide the foundation for new research in the area of concurrent program testing, such as definition of new criteria and new testing tools.

Keywords: software testing, concurrent program, systematic review

1 Introduction

The objective of the concurrent programming is to increase application performance, allowing better use of available resources. A concurrent application consists of several processes that interact to solve a problem, reducing the computational time in several application domains, such as: web services, corporative systems, systems based on Service Oriented Architecture (SOA) and complex systems, such as weather forecast, dynamic molecular simulation, bio-informatics and image processing.

Each day there is a growing demand for concurrent software due to the possibility of making more efficient processing big volumn of the date and advanced of the hardware tecnology that providing ever more efficient machines. The concurrent programming can reduce the computational time in several fields, such as: weather forecasting, fluid dynamics, image processing, quantum chemistry, among others [53]. Previous research have defined techniques and criteria for validation of the traditional programs [40, 23, 27, 43, 8, 55, 36].

For traditional programs, many of the testing problems were reduced with the introduction of testing criteria and the implementation of supporting tools.

* The authors are financially supported by FAPESP and CNPq – Brazil.

A testing criterion is a predicate to be satisfied by a set of test cases and can be used as a guideline for the generation of test data. Structural criteria utilize the code, the implementation and structural aspects of the program to select test cases. They are usually based on control-flow graph and definition and use of variables in the program [43]. Testing criteria allow selecting a subset of the input domain preserving the probability of revealing the existent bugs. These criteria systematize the testing activity and also constitute a coverage measure of this activity [40].

Unlike traditional programs, concurrent programs have specific features such as communication, synchronization and nondeterminism, making more complex the testing activity. During execution, computations in a concurrent program can envolve a high number of the interactions among processes and the number of possible execution paths in the program can be extremely large. In this sense, concurrent program testing must considers both sequential aspects and specific aspects of these programs.

The current demand for distributed applications in domains as web services and corporative systems has increased the interest for concurrent programming. It is observed that the concurrent programming acts as a common-base for these technologies, providing mechanisms for processes management and for interaction among processes. In this way, the testing activity is essential to guarantee the quality and the reliability of these applications. This aspect has motivated the study about bugs patterns and mechanisms of the testing for concurrent and parallel applications.

According to our knowledge this is the first effort to provide a systematic review in context of concurrent programs. A systematic review is a process of assessment and interpretation of all available works related to a research question or subject of specific interest [25], providing a background for further investigation. This technical report describes a systematic review aimed at obtaining evidence of the current state of research related to test criteria, bugs taxonomy and testing tools for concurrent and parallel programs.

The remainder this technical report is structured as follows. In Section 2, testing of concurrent programs is described, presenting studies related to definition of testing in this context. In Section 3 is described the planning and the method used to conduct the systematic review. The results of this review are presented in Section 4. Concluding remarks are presented in Section 5.

2 Testing of the Concurrent Programs

There are two paradigms for interaction among concurrent processes: shared memory and message passing [1]. When message passing is considered, the communication (and synchronization) among concurrent processes occurs by exchanging messages. In case of shared memory, the communication occurs by sharing address spacing and the synchronization occurs by synchronization primitives, such as semaphores or monitors. These differents aspects need to be considered for the proposition of the testing mechanisms for concurrent programs,

considering features, such as non-determinism, synchronization and communication.

Yang [59] describes some challenges to test concurrent programs: 1) developing static analysis; 2) detecting unintentional races and deadlock situations in nondeterministic programs; 3) forcing a path to be executed when nondeterminism might exist; 4) reproducing a test execution using the same input data; 5) generating the control flow graph of nondeterministic programs; 6) providing a testing framework as a theoretical base for applying sequential testing criteria to parallel programs; 7) investigating the applicability of sequential testing criteria to parallel program testing; and 8) defining test coverage criteria based on control and data flow.

Identification of the typical bugs in domain is essential to define testing techniques. In this direction, some studies have defined bugs taxonomy for concurrent programs. Based on bugs taxonomy for sequential programs [22], Krawczyk and Wisniewski [26] extended this taxonomy, presenting two bugs related to concurrent programs: observability and locking bugs. The observability bug is a special kind of domain bug, related to synchronization bug. These bugs can be observed or not during the execution of a same test input; the observation depends on the parallel environment and on execution time (non-determinism). Locking bugs occur when the parallel program does not finish its execution, staying locked, waiting forever.

Some initiatives to define testing criteria for concurrent programs can be found for shared memory parallel programs [5, 54, 58, 60, 57]. Other works have investigated the detection of race conditions [10, 29] and mechanisms to replay testing for nondeterministic programs [7, 9]. However, few studies are found that investigate the application of the testing coverage criteria and supporting tools in context of message passing parallel programs. For these programs, new aspects need to be considered. For instance, data flow information must consider that an association between one variable definition and one use can occur in different addressing spaces. Because of this different paradigm, the investigation of challenges mentioned above, in the context of message-passing parallel programs, is not a trivial task and presents some difficulties.

Yang and Chung [57] introduce the path analysis testing of concurrent programs. Given a program, two models are proposed: 1) task flowgraph, which corresponds to the syntactical view of the task execution behavior and models the task control flow; and 2) rendezvous graph, which corresponds to the runtime view and models the possible rendezvous sequences among tasks. An execution of the program will traverse one concurrent path of the rendezvous graph (C-route) and one concurrent path of the flowgraph (C-path). A method called controlled execution to support the debugging activity of concurrent programs is presented. They pointed out three research issues to be addressed to make practical their approach: C-path selection, test generation and test execution.

Taylor et al. [54] propose a set of structural coverage criteria for concurrent programs based on the notion of concurrent states and on the concurrency graph. Five criteria are defined: all-concurrency-paths, all-proper-cc-histories,

all-edges-between-cc-states, all-cc-states and all-possible-rendezvous. The hierarchy (subsumption relation) among these criteria is analyzed. They stress that every approach based on reachability analysis would be limited in practice by state space explosion. They mentioned some alternatives to overcome the associated constraints.

In the same vein of Taylor and colleagues' work, Chung et al. [5] propose four testing criteria for Ada programs: all-entry-call, all-possible-entry-acceptance, all-entry-call-permutation and all-entry-call-dependency-permutation. These criteria focus the rendezvous among tasks. They also present the hierarchy among these criteria.

Yang et al. [58] extend the data flow criteria [43] to shared memory parallel programs. The parallel program model used consists of multiple threads of control that can be executed simultaneously. A Parallel Program Flow Graph (PPFG) is constructed and is traversed to obtain the paths, variable definitions and uses. All paths that have definition and use of variables related with parallelism of threads constitute test requirements to be exercised. The Della Pasta Tool (Delaware Parallel Software Testing Aid) automates their approach. The authors presented the foundations and theoretical results for structural testing of parallel programs, with definition of the all-du-path and all-uses criteria for shared memory programs.

Yang et al. [58] works inspired the test model definition for message-passing parallel programs, defined in [49]. This test model, called PCFG (Parallel Control Flow Graph), captures statically information about control, data, communication and synchronization of the message-passing parallel programs. Coverage criteria were defined, considering the communication, the data-flow and the control-flow. These criteria allow the testing of sequential and parallel aspects of the concurrent programs. To support the effective application of the testing criteria defined, ValiPar tool was implemented. ValiPar is able to validate concurrents programs in different message-passing environments with a fixed number of processes. It is currently instanced for PVM (Parallel Virtual Machine) and MPI (Message Passing Interface) parallel programs in C language [48, 20].

Based on previous work, the test model was extended for shared memory concurrent programs [44]. The work also presents a post-mortem method based on timestamps [29], to determine which communications (related to shared variables) happened in an execution. This information is important to establish pairs of definition and use of the shared variables. The proposed testing criteria are based on models of control and data flow and include the main features of the most used $PThreads/ANSIC$ programs. The model considers communication, concurrency and synchronization faults among threads and also fault related to sequential aspects of each thread. These ideas were implemented in a testing tool, called ValiPThread. This tool follows the ValiPar architecture, adapting the modules for context of shared memory programs.

In this direction, Endo et al. [11] extend a test model and testing criteria for SOA context, considering web services specified in WS-BPEL. The authors

defined coverage criteria for services composition testing. In the same way, ideas were implemented in a testing tool, called ValiBPEL.

3 Systematic Review: Planning and Conducting

Primary studies[1] described in previous section show some works developed before year 2000. Information about testing criteria, bugs taxonomy and tool related to concurrent programs are presented in disjunct way. Therefore, this review intends to identify new publications, from which relationships can be established between the proposed approaches.

A systematic review is a process of assessment and interpretation of all available works related to a research question or subject of specific interest, providing a background for further investigation [25]. This review intends to identify current publications related to testing criteria, bugs taxonomy and testing tools for concurrent and parallel programs. Another objective is observing the relationship among proposed approaches, facilitating the development of new researhes in this area.

This review was performed according to the process defined by Kitchenham and Charters [25]. This process is composed of three phases: 1) planning - definition of a protocol that specifies the plan that the systematic review will follow, 2) conducting - execution of the protocol planned and 3) reporting - divulgation of the results. The review was performed by two reviewers, being a master student and her advisor. The main points of planning are presented in sequence.

3.1 Planning

Three research questions were formulated to the objectives of the systematic review:

Question 1: What testing criteria were proposed to test concurrent programs? - The interest is to know if exists new works that propose coverage criteria specific for concurrent programs.

Question 2: What type of bugs related to concurrent programs were identified? - typical bugs help in the definition of new coverage criteria. In this question the objective is to find bugs taxonomy for concurrent programs, improving the taxonomy already known.

Question 3: What tools were proposed for concurrent program testing? - testing tools provide support for execution of testing activity. The identification of tools avaliable helps in comparison of the resources that each tool offers.

[1] Primary studies (in the context of evidence) is an empirical study investigating a specific research question [25]

According to systematic review process [25], the question is structured in four aspects (PICO):

• **Population**: papers, dissertations and thesis;
• **Intervention**: software technology, tool or procedure, which generates the outcomes;
• **Comparison**: comparison between the items contained in the intervention;
• **Outcomes**: technology impact in relevant information for classifications of bugs, criteria and testing tools to concurrent program to the profession practic.

So, according to the systematic review process [25], the research questions are structured in four aspects:
• **Population** - refers to execution of test activity. The population is composed of researchers of the software testing and developers;
• **Intervention** - testing criteria, classifications of bugs and testing tools for concurrent program;
• **Comparison** - comparison between classifications of bugs, testing criteria and testing tools for concurrent programs.
• **Outcomes** - this study aims to obtain testing criteria, bugs taxonomy and tools for concurrent program testing.

Search strategy

Based on research questions, some keywords were extracted and used to search the primary study [2] sources. The initial set of keywords was: *"concurrent program", "defect taxonomy" and "test"*. During the preliminary search conducted other keywords were added to improve the search strings.

The terms and synonyms were combined using boolean AND and OR operators. From this combination, the following search string was constructed:

(((multithread) OR (distributed program) OR (distributed software) OR (parallel program) OR (concurrent program) OR (parallel software) OR (concurrent software)) AND (((((taxonomy) OR (model) OR (classification)) AND ((defect) OR (fault) OR (error) OR (bug))) OR ((test) OR (testing))) NOT ((verification) OR (modelling) OR (fault tolerance) OR (fault-tolerance) OR (failure tolerance) OR (network) OR (protocol))))

Four digital libraries were considered for conduction of the searches 1. These libraries were chosen due to they present the most relevant sources about software engineering and distributed systems.

The searches were performed in November, 2009. All the results of the search process were documented, therefore, others research can to acess this documentation and, if necessary, to repeat the search process, for example,

[2] Primary studies (in the context of evidence) is an empirical study investigating a specific research question [25].

Table 1. Digital library selected

Source	Address
ACM Digital Library	(portal.acm.org)
IEEE eXplore	(ieeexplore.ieee.org)
Engineering Village	(engineeringvillage.com)
SCOPUS	(scopus.com)

considering another period for the search.

Studies Selection

The following inclusion criteria (IC) were defined in order to obtain primary studies that could provide answers to research questions:

• **(IC1) Primary studies that present testing criteria for concurrent programs** - the testing criteria selected must be specific for concurrent programs. If the testing criteria are adapted from sequential programs, these might still be selected.

• **(IC2) Primary studies that characterize specific bugs related to concurrent programs** - the type of bugs defined by works need to be specific for concurrent programs.

• **(IC3) Primary studies that propose tools for support concurrent program testing** - the tools found should give suport for testing activity of concurrent programs.

Not all of these inclusion criteria must be presented to every primary study, so if only one criterion is present in a primary study, this must be included.

The following exclusion criteria (EC) were defined to eliminate primay studies that are not related to search questions:

• **(EC1) primary studies that present testing approaches not related to concurrent programs** - primary study that do not presents approaches (testing criteria and techniques) defined for concurrent programs will be excluded.

• **(EC2) primary studies that present type of bugs not related to concurrent programs** - primary studies that do not characterize type of bugs specific of concurrent programs will be excluded.

• **(EC3) primary studies that proposes testing tools not related to concurrent program** - primary studies that present tools or parts of tools not related to concurrent program testing will be excluded.

Visual Text Mining (VTM) technique [39] was used to support the studies selection. VTM uses text mining algorithms and methods combined with interactive visualizations to help the user making sense of a collection of documents, without actually reading all of them. In this case the studies were reading partially or full. So this approach helped in the review of the studies.

Data Extraction

In data extraction stage is defined forms for accurately record the information obtained by researchers from the primary studies [25]. The data extraction form provides some standard information such as: font of extraction, title, authors and list of conclusion found for each research question. The procedure for data extraction occurred after of the selection of the studies. For each study analysed was written one resume to facilitate the documentation of the responses for the research questions.

Quality Assessment

After the data extraction we realize the quality assessment of the studies. A checklist for quality assessment was made for estimate the quality of the primary study selected. The quality criteria (QC) were defined using general information of the studies, including the following questions: (C1) presentation of the goals, (C2) presentation of ideas, (C3) contribution to research area and (C4) applicability of the proposal. The primary studies analysed through this cheklist were scored as follows:

• **Data presented are incomprehensible or without implementing the proposal**: 0.0 point;
• **Data presented with understanding difficult or that includes presentation of case study or examples**: 0.5 point;
• **Data presented with understanding easy or that includes presentation of experimental study**: 1.0 point.

3.2 Conduction

After the definition of search string general, the searches were performed in digital libraries choosen. In the phase of preliminary selection a reading of abstracts of each primary study was done and, when necessary, a reading of the conclusion or the introduction of the study. The studies that not indicated sufficient evidence for inclusion or exclusion were discussed by reviewers. The search results in this phase were tabulated in: primay studies by font, studies included and excluded.

In the phase of final selection, the studies were analyzed completely and the data extracted were resumed with the purpose of documentation and to facilitate the discussions during slection phase. After selection of the studies, they were validated using VTM technique and after this activity the data were extracted.

The data were analysed to obtain conclusions about similarities and differences among the approachs proposed.

In Table 2 is presented the number of the studies selected in each source and in each phase of the study (preliminary and final selection). The duplicate studies returned were excluded. The search returned 305 studies; after of extraction of the duplicate studies and not relevant data (for example, proceedings) 261 studies were obtained. After included and excluded criteria remained 44 relevants studies. In next phase (reading complete of each study), was not possible access 2 studies [6, 19], that were excluded of the review. With the review of complete studies were selected 14 primary studies.

Table 2. Number of studies selected in each source

Source	Return	Phase 1		Phase 2	
		Included	**Excluded**	**Included**	**Excluded**
ACM	83	13	70	06	07
Engineering Village	75	07	68	02	05
IEEE	34	09	25		09
SCOPUS	70	15	55	06	10

3.3 Validation

In validation stage was used an approach that uses VTM technique and the associated tool - *Projection Explorer (PEx)* - to support the inclusion and exclusion decisions [14]. The *PEx* [42] provides several features for text manipulation and exploration of documents collections using VTM techniques. In summary, the original PEx adopts a two-dimensional visual representation, known as document map, where each document is mapped to a graphical element on the plane, usually a circle (point), with points' relative positions reflecting similarity relationships between the contents of the documents they represent. Thereby, on the layout, similar documents are put close to one another, while dissimilar ones are supposed to be positioned far apart.

Figure 1 presents a document map generated using PEx. This map is composed of 261 primary studies analyzed in this systematic review, highlighting them using different colors in order to differentiate in which of the stages a study was removed from the review. White points are studies excluded in first stage (just reading the abstract), gray points are the studies excluded in second stage (reading the full article) and the black points are the included.

Acording to Felizardo et al. [14] the exploration of a document map is conducted in two steps: (1) first, a clustering algorithm is applied to the document map, creating groups of highly related (similar) documents; (2) then, the resulting clusters are analyzed in terms of: **Pure Clusters** – all documents belonging to a cluster have the same classification (all included or excluded, regardless of exclusion stage). Normally, such cases do not need to be reviewed; and **Mixed Clusters** – documents with different classifications on the same cluster. These

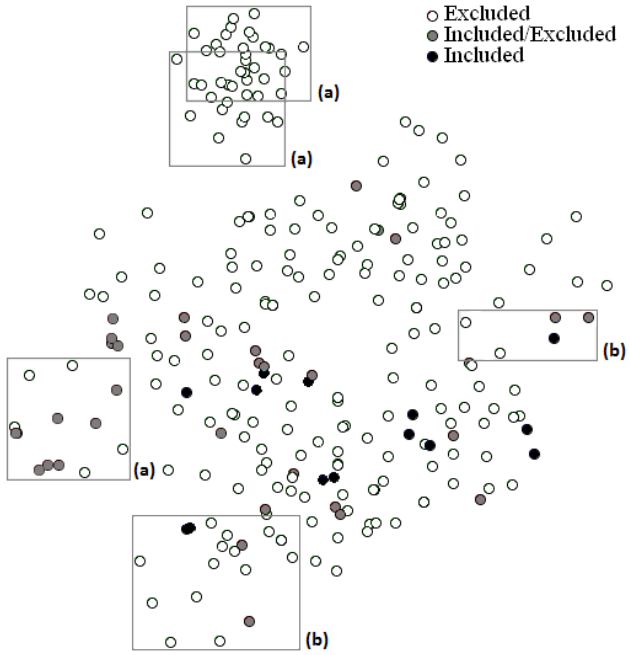

Fig. 1. Document map colored with the history of the inclusions and exclusions of the studies.

cases are hints to the reviewer, and the studies grouped there should be reviewed following the traditional method.

To facilite the visualization, it is represented in Figure 1 just some of the clusters generated by PEx. Examples of pure clusters (all excluded) are identified in Figure 1 using the label (a) and therefore do not needed to be reviewed. Mixed clusters (clusters containing black (included) and white or gray (excluded) studies) are identified using the label (b) and they were reviewed by us. At the end, we kept the initial classifications conducted manually, but this technique contributed to a review of studies that could have been wrongly excluded or included previously.

Quality assessment of studies The Table 3 provides information on scores and comments from the evaluation of quality of the studies included in the review (based on checklist defined in Section 3.1. Each study can be identified through the column 1 that show the references. In column 2 to column 5 are described the scores of the studies. In column 6 are presented information about each study that help the understanding of the score assigned for them.

The studies were analysed on the scale of 0 to 1 considering aspects how clarity in the presentation of the goals of the work and approach presented, contribution the area of research and application of the proposed.

Table 3. Quality assessment of study

Reference	C1	C2	C3	C4	Comment
[32]	1	0.5	1	0.5	Presents surface ideas in some sections and examples to explain the approach
[50]	1	1	1	1	Presents a case study
[60]	1	1	1	1	Presents a case study
[44]	1	1	1	0.5	Presents a case study
[45]	1	1	1	1	Presents a case study
[37]	1	0.5	1	0.5	Approach is not presented clarity
[3]	1	1	1	1	Presents a case study
[34]	1	0.5	1	0.5	Approach succinctly presented using illustrative figure
[33]	1	1	1	1	The bug reported were based on real study
[52]	1	1	1	0.5	Presents a case study
[38]	1	1	1	0.5	Presents a real application of the approach presented
[35]	1	1	1	1	The bug reported were based on real study
[41]	1	1	1	1	Presents a case study
[17]	1	1	1	1	Presents a case study

4 Results

In Table 4 the selected studies are grouped by each inclusion criteria. In second column are presented the testing technique considered for each study. In third column are presented the references of the each study.

Table 4. Selected studies

Inclusion Criteria	Technique	Reference
Presents testing criteria	Structural testing	[32, 60, 44, 34, 52]
	Error based testing	[3]
	Model based testing	[45, 37, 38]
Presents bug taxonomy		[3, 33, 35]
Presents testing tool	Structural testing	[50, 41, 17]

4.1 Research questions result

In this section are presented the answers for each research questions defined in Section 3.1.

What testing criteria were proposed to test concurrent programs? In context of structural testing were obtained 5 studies. Ling et al. [32] present a testing criterion for message passing concurrent programs that exercises SYN-sequences. A SYN-sequence is composed of series of ordered SYN-events (send or receive primitives). The conformance testing can help determine the test sequences for the criterion. There are many limitations in the testing of practical distributed programs using this criterion. Lu et al. [34] explore this approach in context of shared memory concurrent programs adding the concept of representative sequences. Representative test sequence is a subset of all possible interleaving of the concurrent events that define the concurrent program behavior. The criteria defined in this work consideres differents features of interleavings and a abstraction level more hight, different of the [32].

Yet related to shared memory there are 3 studies. Yang and Pollock [60] present all-uses criterion, a test model and the challenges related to concurrent program testing. This work present this criterion in detail, different of the work [34] that present the criteria superficially, exploring access to reading and writing of variables.

Takahashi et al. [52] propose a model based approach and define the criteria ACP (all-concurrent-path) and ACBP (all-concurrent-binomial-path). This approach is different from previous studies because this approach uses the concept of the abstract block. For this, it considers hardware and low level of commands in order to reduce the number of combinations between statements of the concurrent program that need to be tested. The authors claim that this criteria can to find specific defects concurrent.

Sarmanho et al. [44] present a family of structural testing criteria considering multithread programs based on semaphores. These criterion allow the testing of sequential and parallel aspects of the concurrent programs. The authors present more information about the criteria all-uses than [60], exploring differents types of associations between definition and use of the variables.

More information about structural testing criteria are presented in Table 5.

In context of model based testing were identified 3 studies (Table 6). Seo et al. [45] propose the use of specification based testing for generation of the representative test sequence. The concurrent program is modeled using Statecharts for generates representative sequences that after are exercised in the automata. The main advantage of approach is that the use of automata may often limit the state explosion problem.

Robinson-Mallett et al. [37] focus on the comunication testing between components of the concurrent program, defining comunication coverage criteria based on model checking. Similarly Seo et al. [45] that seeked greater effectiveness in the test by to select representative sequences, this work also explores this aspect, but considering that the specification is given in open interface automata. The advantage of this approach is the generation of a potentially minimal set of test cases. Extending this work, [38] takes advantage of the testability of communication between components for the integration testing.

Table 5. Structural testing criteria for concurrent programs

Testing Criteria	Paradigm	Test Model	Funcionality
Ling et al. [32]	message passing	synchronization graph	Define criteria to validate SYN-sequence events. One SYN-sequence is valid if satisfies the program specification.
Interleaving coverage [34]	shared memory	based on concurrent faults	Define criteria to exercise synchronization sequence of the concurrent program.
Du-Pair [60]	shared memory	PPFG (parallel graph)	Define criterion to exercise pairs definition-use of variables in shared-memory concurrent programs.
ACP and ACBP [52]	shared memory	CMFG (Concurrent Module Flow Graph)	The criteria exercise all paths of the concurrent program.
Data, control and comunication flow [44]	shared memory	PCFG_SM (parallel graph)	The criteria exercise sequential and specific features of the shemaphore based programs.

Table 6. Model based testing for concurrent programs

Testing Criteria	Paradigm	Test Model	Funtionality
Seo et al.[45]		statechart	Aims to exercise synchronization sequence.
Sr-pairs and sr-paths[37]		timed interface automaton and CTL	Exercise each pair of the send/receive and paths related to communication between process.
Sender, receiver, sr-pairs e sr-paths [38]		timed interface automaton and CTL	Criteria defined in previous work [37] are extended to consider timed interface automaton. Timed interface automaton models the system interface, considering temporal aspects related to states of automaton.

In context of error based testing was obtained one study that propose a mutation operator set for shared memory using *Java* programs [3]. The objective is modeling bugs related to aspects of the concurrency and synchronization of this language. The operators were definided based on concurrent faults related to concurrent aspects, such as: concurrent method, calls of the concurrent method and critical region.

What type of bugs related to concurrent programs were identified? The main aspects of bugs taxonomy of the concurrent programs are presented in Table 7.

In the context of bugs taxonomy were obtained 3 studies. Farchi et al. [13] present a set of bugs pattern not programming language specific. In this work the authors show deduction of concurrent bug patterns from design patterns. The taxonomy presented consideres execution of comands of language in low level and comands for concurrency of the language for definition of possible

Table 7. Classification of bugs for concurrent programs

Classification	Language	Application	Features
Bradbury	Java	shared memory	bugs related to concurrent aspects of *Java* programs were included in taxonomy proposed by Farchi et al. [13].
Loureno and Cunha	C	Software Transactional Memory	bugs related to interleaving sequences in STM (transactional memory) and problems related to data update.
Lu et al.	C/C++		Two types of concurrency bugs are presented: deadlock bugs (31) and non-deadlock concurrency bugs (74). The non-deadlock bugs were classified in: atomicity-violation bugs, order-violation bugs and other bugs based on other causes.

bugs. Based on this classification, Bradbury [3] analysed the IBM Concurrency Benchmark [12] and defined bugs in context of real application to *Java*. In this classification are added aspects that not available when the taxonomy in [13] was proposed, such as other signals notify that can are missing, problems with starvation or number of resources for threads, or incorrect count initialization.

In Loureno and Cunha [33] are present bugs set based on Software Transactional Memory (STM). STM is a control mechanism similar to competitive transactions of databases, but in this case to control access to shared memory in concurrent computing [47]. In this work are considered to definition of the bugs features of STM, such as read set not consistent, read/write variables, lost update on lock upgrade, betweeen other.

Lu et al. [35] present a study of the concurrents bugs in real application, using 4 open source application developed in C or $C++$ (MySQL, Apache, Mozilla and OpenOffice). This classification considers analysis of higher level of that the classification of [13], when considers threads number, of variables and acess involved in this bugs. But the biggest difference among this bugs and the classification of Farchi et al. [13] is that they consider real application and real bugs.

What tools were proposed for concurrent program testing? Were found 3 testing tool for shared memory and language *Java*. Stoller [50] presents the *rstest* tool (random scheduling test), that was implemented using bytecode transformation, inserting calls to a scheduling function at selected points of the program. This tool is similar the tool described by Joshi et al. [17] named *CalFuzzer* for active testing of the *Java* programs, because the two use scheduling.

CalFuzzer works with the concept of active testing. Active testing works in two phases, first uses predictive off-the-shelf static or dynamic program analyses to identify potential concurrent bugs. In the second phase, active testing uses the reports from these predictive analyses to explicitly control the underlying scheduler of the concurrent program to accurately and quickly discover real concurrent bugs. *CalFuzzer* uses one randomic thread scheduling and uses a predictive program analysis that fornece information about potetial concurrency bugs. The tool possibility to detect data races, atomicity violations and dead-

locks. The disadvantage of *CalFuzzer* is that it control execution by scheduling one thread at a time, leaving out the advantages of multicore machine in testing.

Based in stress testing, Park et al. [41] proposed the *CTrigger* tool to identify bugs related to atomicity violation. Similarly the *CalFuzzer* that works in two phases, the *CTrigger* also includes two phases: 1) trace analysis to obtain a list of unserializable interleavings and 2) analysis of these unserializable interleavings using controlled testing to find and atomicity violation. *CTrigger* controls execution by suspendig a threads execution at appropriate places to increase the occurrence probability of the targeting unserializable interleaving.

4.2 Additional information

One class of the studies observed in the second phase of the review present techniques to treat the nondeterminism, using controled execution [18, 46, 51, 31, 28, 56, 24, 16, 30, 4]. Those studies were excluded of the review, because the proposed approaches not were related with the research questions defined to the review.

5 Conclusion

This technical report presented a systematic review aiming to find out research on concurrent software testing. As result, the review identified testing criteria, bugs taxonomy and testing tools that can provide foundation for new research in the area of concurrent software testing.

The systematic review includes studies related to three research questions defined. Most studies present definition of testing criteria for concurrent programs. Basically, testing criteria found are related to structural, model based and based error testing techniques. These criteria explore comunication inter process, components of the concurrent programs and message passing, in most cases, considering shared-memory programs. It was observed that 64.3% of the studies present testing criteria. In this studies, 55.5% are structural testing criteria, 33.3% are model based testing criteria and 11.1% present error based testing criteria. Considering bugs taxonomy, 21.4% of the studies present definition of the typical bugs for concurrent programs. Of the studies 21.4% present testing tool.

Through the results obtained in this review, it is observed a lack of the testing criteria and tools for concurrent programs, considering intrinsic aspects of these programs. We plan realize a review more complete, considering in the research string all aspects of concurrent programs. Also, there is a lack of experimental studies to evaluate testing criteria and bugs taxonomy. Experimental studies are fundamental to provide information about application cost, efficacy and complementary aspect of the testing criteria.

Thus, we plan the development of a experimental study, using bug taxonomy of the primary studies, to evaluate of the efficacy in fault reveal of the testing criteria for concurrent programs. We hope this study will contribute to

the definition and the evolution of testing strategies in context of concurrent programs.

References

1. Almasi, G. S. and Gottlieb, A.: Highly Parallel Computing. The Benjamin Cummings Publishing Company, Redwood City, CA, USA (1994)
2. Bertolino, A.: Software testing research: Achievements, challenges, dreams. In Internation Conference on Software Engineering, pages 85103. IEEE Computer Society (2007)
3. Bradbury, J. S. Using program mutation for the empirical assessment of fault detection techniques: a comparison of concurrency testing and model checking. PhD thesis, Kingston, Ont., Canada (2007)
4. Chen, J. and MacDonald, S.: Towards a better collaboration of static and dynamic analyses for testing concurrent programs. In: PADTAD '08: Proceedings of the 6th workshop on Parallel and distributed systems, pp. 1–9. ACM, New York, NY, USA (2008)
5. Chung, C-M. and Shih, T. K. and Wang, Y-H. and Lin, W-C. and Kou, Y-F.: Task Decomposition Testing and Metrics for Concurrent Programs. In: Fifth ISSRE, pp. 122–130 (1996)
6. Chung, I. and Kim, B.: A new approach to deterministic execution testing for concurrent programs. IEICE Transactions on Information and Systems. E84-D, 12, 1756–1766 (2001)
7. Carver, R. H. and Tai, K. C.: Replay and Testing for Concurrent Programs. IEEE Software. pages 74–86 (1991)
8. Coward, P. D.: A review of software testing. Inf. Softw. Technol. 30, 3, 189–198 (1988)
9. Damodaran-Kamal, S. K. and Francioni, J. M.: Nondeterminacy: Testing and Debugging in Message Passing Parallel Programs. In: 3rd ACM/ONR Workshop on Parallel and Distributed Debugging, pp. 118–128. ACM Press, (1993)
10. Edelstein, O., Farchi, E., Goldin, E., Nir, Y. Ratsaby, G. and S. Ur.: Framework for testing multi-threaded Java programs. Concurrency and Computation: Practice and Experience, 15, 3-5, pp. 485–499 (2003)
11. Endo, A. T., Simão, A. S., Souza, S. R. S. and Souza, P. S. L.: Web Services Composition Testing: A Strategy Based on Structural Testing of Parallel Programs. In: TaicPart: Testing Academic & Industrial Conference - Practice and Research Techniques. pp. 3–12. IEEE Computer Society, Washington, DC, USA (2008)
12. Eytani, Y. and Ur, S.: Compiling a benchmark of documented multi-threaded errors. Parallel and Distributed Processing Symposium, International, 17, 266 (2004)
13. Farchi, E., Nir, Y., and Ur, S.: Concurrent errors patterns and how to test them. In: IPDPS '03: Proceedings of the 17th International Symposium on Parallel and Distributed Processing, Washington, DC, USA (2003)
14. Felizardo, K. R. and Maldonado, J. C.: A visual approach to assist the selection review of primary studies in systematic reviews. In: VI Experimental Software Engineering Latin American Workshop (ESELAW'09), pp. 83–92 (2009)
15. Fraser, G., Wotawa, F., Ammann, P.E.: Testing with model checkers: A survey. Software Testing Verification and Reliability. 19, 3, pp. 215–261 (2009)
16. G, X., Wang, Y., Zhou, Y., and Li, B.: On testing multi-threaded java programs. In: Haier International Training Center. 1, pp. 702 –706. IEEE Computer Society, Los Alamitos, CA, USA (2007)

17. Joshi, P., Naik, M., Park, C.-S., and Sen, K.: Calfuzzer: An extensible active testing framework for concurrent programs. In: CAV '09: Proceedings of the 21st International Conference on Computer Aided Verification. pp. 675–681, Berlin, Heidelberg (2009)
18. Harvey, C. and Strooper, P.: Testing java monitors through deterministic execution. In: ASWEC '01: Proceedings of the 13th Australian Conference on Software Engineering, pp. 61. IEEE Computer Society, Washington, DC, USA (2001)
19. Hashimoto, K., Tsuchiya, T., and Kikuno, T.: Error models and fault-secure scheduling in multiprocessor systems. IEICE Transactions on Information and Systems. E84-D, 5, 635–650 (2001)
20. Hausen, A. C., Vergilio, S. R., Souza, S. R. S., Souza, P. S. L. and Simo, A. S.: A Tool for Structural Testing of MPI Programs. In: LAtin-American Test Workshop - LATW, pp. 1–1. 8th IEEE LAtin-American Test Workshop, Cuzco, Peru (2007)
21. Hierons, R.M., Bogdanov, K., Bowen, J.P., Cleaveland, R., Derrick, J., Dick, J., Gheorghe, M., Harman, M., Kapoor, K., Krause, P., Luttgen, G., Simons, A.J.H., Vilkomir, S., Woodward, M.R., Zedan, H.: Using formal specifications to support testing. ACM Computing Surveys. 41, 2 (2009)
22. Howden, W. E.: Reliability of the path analysis testing strategy. IEEE Transactions on Software Engineering. 2, 208–215 (1976)
23. Howden, W. E.: Methodology for the generation of program test data. IEEE Trans. Comput. 24, 5, 554–560 (1975)
24. Kilgore, R. B.: Testing concurrent software systems. PhD thesis. University of Texas at Austin, Austin, TX, USA (2006)
25. Kitchenham, B. and Charters, S.: Guidelines for performing systematic literature reviews in software engineering. Technical Report EBSE 2007-001, Keele University and Durham University Joint Report (2007)
26. Krawczyk, H. and Wiszniewski, B.: Classification of Software Defects in Parallel Programs. Technical report, Faculty of Electronics, Technical University of Gdansk, Poland, 2 (1994)
27. Laski, J. W. and Korel, B.: A data flow oriented program testing strategy. IEEE Trans. Softw. Eng. 9, 3, 347–354 (1983)
28. Lei, Y. and C. R.: Reachability testing of semaphore-based programs. In: COMPSAC '04: Proceedings of the 28th Annual International Computer Software and Applications Conference. pp. 312–317. IEEE Computer Society, Washington, DC, USA (2004)
29. Lei, Y. and Carver, R. H.: Reachability Testing of Concurrent Programs. IEEE Transactions on Software Engineering, 32, 6, 382–403 (2006)
30. Lei, Y., Carver, R. H., Kacker, R., and Kung, D.: A combinatorial testing strategy for concurrent programs. In: Softw. Test. Verif. Reliab. 17, 4, pp. 207–225. John Wiley and Sons Ltd, Chichester, UK (2007)
31. Li, S. Q., Chen, H. Y., Yu, X. S.: A framework of reachability testing for java multithread programs. In: IEEE International Conference on Systems, Man and Cybernetics. 3, pp. 2730 – 2734. IEEE Press (2004)
32. Liang, Y., Li, S., Zhang, H., and Han, C.: Timing-sequence testing of parallel programs. Journal of Computer Science and Technology. 15, 1, 84 – 95 (2000)
33. Lourenço, J. and Cunha, G.: Testing patterns for software transactional memory engines. In: PADTAD '07: Proceedings of the 2007 ACM workshop on Parallel and distributed systems: testing and debugging. pp. 36–42, New York, USA (2007)
34. Lu, S., Jiang, W., and Zhou, Y.: A study of interleaving coverage criteria. In: ESEC-FSE '07: Proceedings of the the 6th joint meeting of the European software

engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering. pp. 533–536, New York, USA (2007)
35. Lu, S., Park, S., Seo, E., and Zhou, Y.: Learning from mistakes: a comprehensive study on real world concurrency bug characteristics. In: ASPLOS XIII: Proceedings of the 13th international conference on Architectural support for programming languages and operating systems, pp. 329–339, New York, NY, USA (2008)
36. Maldonado, J. C.: Critérios Potenciais Usos: Uma Contribuição ao Teste Estrutural de Software. Tese de doutoramento, CA/FEE/UNICAMP, Campinas, SP (1991)
37. Robinson-Mallett, C., Hierons, R. M., and Liggesmeyer, P.: Achieving communication coverage in testing. SIGSOFT Softw. Eng. Notes. 31, 6, 1–10 (2006)
38. Robinson-Mallett, C., Hierons, R. M., Poore, J., and Liggesmeyer, P.: Using communication coverage criteria and partial model generation to assist software integration testing. Software Quality Control. 16, 2, 185–211 (2008)
39. Malheiros, V., Hohn, E., Pinho, R., Mendonca, M. and Maldonado, J. C.: A Visual Text Mining approach for Systematic Reviews. In: ESEM '07: Proceedings of the First International Symposium on Empirical Software Engineering and Measurement, pp. 245–254. IEEE Computer Society, Washington, DC, USA (2007)
40. Myers, G. J., Sandler, C., Badgett, T., and Thomas, T. M.: The Art of Software Testing. John Wiley & Sons, Inc., Hoboken, New Jersey (2004)
41. Park, S., Lu, S., and Zhou, Y.: Ctrigger: exposing atomicity violation bugs from their hiding places. In: ASPLOS '09: Proceeding of the 14th international conference on Architectural support for programming languages and operating systems, pp. 25–36, New York, NY, USA (2009)
42. Paulovich, F. V., Oliveira, M. C. F., and Minghim, R.: The projection explorer: A flexible tool for projection-based multidimensional visualization. pp. 27–36 (2007)
43. Rapps, S. and Weyuker, E. J.: Selecting Software Test Data Using Data Flow Information. IEEE Transaction Software Engineering. 11, 4, 367–375 (1985)
44. Sarmanho, F. S., Souza, P. S., Souza, S. R., and Simão, A. S.: Structural testing for semaphore-based multithread programs. In: ICCS '08: Proceedings of the 8th international conference on Computational Science, Lecture Notes in Computer Science, pp. 337–346. Springer-Verlag, Krakow (2008)
45. Seo, H.-S., Chung, I. S., and Kwon, Y. R.: Generating test sequences from statecharts for concurrent program testing. IEICE - Trans. Inf. Syst. E89-D, 4, 1459–1469 (2006)
46. Seo, H.-S., Chung, I. S., Kim, B. M., and Kwon, Y. R.: The design and implementation of automata-based testing environment for java multi-thread programs. In: APSEC '01: Proceedings of the Eighth Asia-Pacific on Software Engineering Conference, pp. 221–228. IEEE Computer Society, Washington, DC, USA (2001)
47. Shavit, N. and Touitou, D.: Software transactional memory. In: PODC '95: Proceedings of the fourteenth annual ACM symposium on Principles of distributed computing, pp. 204–213. New York (1995)
48. Souza, S. R. S., Vergilio, S. R., Souza, P. S. L. Simão, A. S., Bliscosque, T. G., Lima, A. M. and Hausen, A. C.: ValiPar: A Testing Tool for Message-Passing Parallel Programs. In: International Conference on Software knowledge and Software Engineering (SEKE05), pp. 386–391. IEEE Press, New York (2005)
49. Souza, S. R. S., Vergilio, S. R., Souza, P. S. L., Simão, A. S. and Hausen, A. C.: Structural testing criteria for message-passing parallel programs. Concurrency Computation Practice and Experience. 20, 16, 1893–1916 (2008)
50. Stoller, S.: Testing concurrent java programs using randomized scheduling. Electronic Notes in Theoretical Computer Science. 70, 4, 149–164 (2002)

51. Tai, K.-C. and Karacali, B.: On godefroid's state-less search technique for testing concurrent programs. In: ISADS '01: Proceedings of the Fifth International Symposium on Autonomous Decentralized Systems. pp. 77–84. IEEE Computer Society, Washington, DC, USA (2001)
52. Takahashi, J., Kojima, H., and Furukawa, Z.: Coverage based testing for concurrent software. In: ICDCSW '08: Proceedings of the 2008 The 28th International Conference on Distributed Computing Systems Workshops, pp. 533–538, Washington, USA (2008)
53. Tanenbaum, A. S.: Modern operating systems. Prentice Hall, Upper Saddle River, NJ (2001)
54. Taylor, R. N. and Levine, D. L. and Kelly, C.: Structural Testing of Concurrent Programs. IEEE Transaction Software Engineering. 18, 3, 206–215 (1992)
55. Ural, H. and Yang, B.: A structural test selection criterion. Inf. Process. Lett. 28, 3, 157–163 (1988)
56. Wildman, L., Long, B., and Strooper, P.: Dealing with non-determinism in testing concurrent java components. In: APSEC '05: Proceedings of the 12th Asia-Pacific Software Engineering Conference, pp. 393–400. IEEE Computer Society, Washington, DC, USA (2005)
57. Yang, R-D. and Chung, C-G.: Path Analysis Testing of Concurrent Programs. Information and Software Technology. 34, 1, 43–56 (1992)
58. Yang, C-S. and Souter, A. L. and Pollock, L. L.: All-Du-Path Coverage for Parallel Programs. In: ISSTA '98: Proceedings of the 1998 ACM SIGSOFT international symposium on Software testing and analysis, pp. 153–162. ACM-Software Engineering Notes, New York, NY, USA (1998)
59. Yang, C-S. D.: Structural Testing of Shared Memory Parallel Programs. PhD thesis, University of Delaware (1999)
60. Yang, C.-S. and Pollock, L.: All-uses testing of shared memory parallel programs. Software Testing Verification and Reliability. 13, 1, 3–24 (2003)