Software Qual J
DOI 10.1007/s11219-015-9301-7

10 Years of research on debugging concurrent and multicore software: a systematic mapping study

Sara Abbaspour Asadollah[1] · Daniel Sundmark[1] · Sigrid Eldh[1,2] · Hans Hansson[1] · Wasif Afzal[1]

Abstract Debugging—the process of identifying, localizing and fixing bugs—is a key activity in software development. Due to issues such as non-determinism and difficulties of reproducing failures, debugging concurrent software is significantly more challenging than debugging sequential software. A number of methods, models and tools for debugging concurrent and multicore software have been proposed, but the body of work partially lacks a common terminology and a more recent view of the problems to solve. This suggests the need for a classification, and an up-to-date comprehensive overview of the area. This paper presents the results of a systematic mapping study in the field of debugging of concurrent and multicore software in the last decade (2005–2014). The study is guided by two objectives: (1) to summarize the recent publication trends and (2) to clarify current research gaps in the field. Through a multi-stage selection process, we identified 145 relevant papers. Based on these, we summarize the publication trend in the field by showing distribution of publications with respect to *year*, *publication venues*, *representation of academia and industry*, and *active research institutes*. We also identify research gaps in the field based on attributes such as *types of concurrency bugs*, *types of debugging processes*, *types of research* and *research contributions*. The main observations from the study are that during the years 2005–2014: (1) there is no focal conference or

✉ Sara Abbaspour Asadollah
sara.abbaspour@mdh.se

Daniel Sundmark
daniel.sundmark@mdh.se

Sigrid Eldh
sigrid.eldh@ericsson.com

Hans Hansson
hans.hansson@mdh.se

Wasif Afzal
wasif.afzal@mdh.se

[1] Mälardalen University, Västerås, Sweden

[2] Ericsson AB, Kista, Sweden

Published online: 21 January 2016

venue to publish papers in this area; hence, a large variety of conferences and journal venues (90) are used to publish relevant papers in this area; (2) in terms of publication contribution, academia was more active in this area than industry; (3) most publications in the field address the data race bug; (4) bug identification is the most common stage of debugging addressed by articles in the period; (5) there are six types of research approaches found, with solution proposals being the most common one; and (6) the published papers essentially focus on four different types of contributions, with "methods" being the most common type. We can further conclude that there are still quite a number of aspects that are not sufficiently covered in the field, most notably including (1) *exploring correction* and *fixing bugs* in terms of debugging process; (2) *order violation*, *suspension* and *starvation* in terms of concurrency bugs; (3) *validation* and *evaluation research* in the matter of research type; (4) *metric* in terms of research contribution. It is clear that the concurrent, parallel and multicore software community needs broader studies in debugging. This systematic mapping study can help direct such efforts.

Keywords Concurrent · Parallel · Multicore · Debugging process · Bugs · Fault · Failure · Systematic mapping study

1 Introduction

The need of maintenance and debugging support for concurrent and multicore software is steadily increasing. Due to the fast growth of systems and applications, there are constantly new demands to adapt to the latest execution paradigms provided by parallelism via multicore and many-core processors. Hardware and system providers (e.g., Intel and IBM), steadily increases the number of cores used. The effort is driven by the need to increase the performance and decrease the power consumption for the new processor generations. The impact and adaptation of software to these new systems causes earlier solved, or partially solved, problems to be re-visited in a new, concurrent, context. Examples of these areas are, e.g., finding the right abstractions of code, new development paradigms and languages, testing for specific multicore patterns and debugging parallel events. In particular, problems in parallel and concurrent execution, e.g., deadlock and race conditions that are resulting in very intricate bugs, lead to increased difficulty. Diagnosing such problems increases the complexity of simultaneously tracing the interactions between the tasks when they access shared resources.

Increased core interaction and managing shared resources may reduce the ability to reuse traditional debugging methods and tools. According to (Petersen et al. 2008), a systematic mapping study helps to provide an overview of a research area by providing a classification, presenting a systematic analysis and highlighting publication venues. The value is twofold: First, new researchers can get a consolidated view of the research, area and second, a research mapping can highlight possibilities for future research.

Our main goal is to identify and analyze work published during the last decade (2005–2014) in the field of debugging concurrent and multicore software. We achieve this goal by fulfilling two objectives: (1) to summarize the recent publication trends and (2) to provide a discussion on current research gaps in the field. We have systematically reviewed related scientific references in the field. We mapped topics studied to existing classifications relevant for the area.

There exists a recent literature review on concurrent software testing (Brito et al. 2010). Their main goal was to obtain evidence of current state-of-research related to testing criteria, testing tools and a bug taxonomy for concurrent and parallel programs. They further provided a list of relevant studies as a foundation for new research in the area. The authors concluded that there is a lack of testing criteria and tools for concurrent programs. They noticed that most experimental studies were providing information on application cost, efficiency and complementary aspects, while there is lack of knowledge on bug taxonomy and on evaluating testing criteria.

Another interesting and relevant study is by Jyoti and Arora (Jyoti and Arora 2014). They presented a survey of debugging and visualization techniques for multi-threaded programs. They categorized, compared and analyzed tools and techniques detecting deadlock, race condition, livelock and other multi-threaded programs bugs. They further presented a list of tools for visualizing multi-threaded programs. They concluded that among tools, the issue of wait-notify anomalies and deadlocks are not considered for debugging visualization.

The emphasis of these two studies (Brito et al. 2010; Jyoti and Arora 2014) is on concurrent, parallel and multi-threaded programs; their focus is not targeting the debugging field. While in this paper, we aggregate the evidence on debugging of concurrent, multi-core and many-core software during the past decade.

The remaining part of this paper is organized as follow: We describe our research method in Sect. 2. In Sect. 3 we present the classification schemes of the study. We provide a comprehensive set of results and the main findings in Sect. 4. The threats to validity are discussed in Sect. 5. In Sect. 6 we discuss our findings. Finally, we conclude the study and highlight the direction of future work in Sect. 7.

2 Research method

This systematic mapping study is carried out by following the guidelines and process proposed by Petersen et al. (Petersen et al. 2008). The process is outlined in Fig. 1. The following subsections describe the steps of this process.

2.1 Definition of research questions (Step 1)

As mentioned in Sect. 1, the overall goal of this study is to identify and analyze work published during the last decade in the field of debugging concurrent and multicore

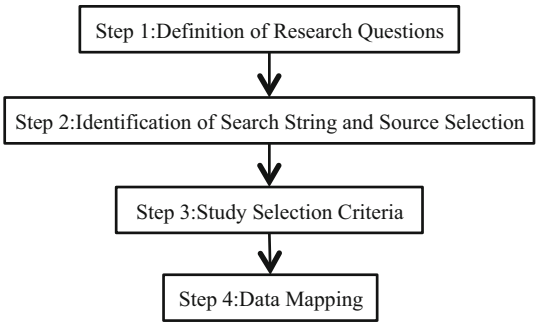


Fig. 1 Workflow of our research process

software. As part of analyzing the published work, we also aim to identify research gaps in the field. We identify research gaps by highlighting topics where research and research results are lacking. We answer the following research questions in this study guided by our objectives given in Sect. 1:

RQ1: What are the publication trends on debugging of concurrent and multicore software during the last decade (between 2005 and 2014)?
RQ1.1: What is the annual number of publications in the field?
RQ1.2: Which venues (e.g., journals and conferences) are the main venues of publication in the field?
RQ1.3: What is the distribution of publications in terms of academic and industrial affiliation? RQ1.4: Which institutes are most active in the field based on number of publications?
RQ2: What are the existing research gaps related to concurrent and multicore software debugging based on:
RQ2.1: What types of concurrency bugs have been addressed?
RQ2.2: What types of debugging processes have been used?
RQ2.3: What types of research are conducted in the area?
RQ2.4: What types of research contributions are presented in the papers?

Answering the above questions provides the main results of our article.

2.2 Identification of search string and source selection (Step 2)

In order to find the possible relevant references, we first identified the most common scientific online digital libraries such as: Web of Knowledge, Scopus and IEEE Xplore. We avoided ACM digital library since it is not very suitable for systematic mapping and literature review studies (Keele 2007) because of the huge number of search results and difficulties in downloading citations to standard bibliographical management tools. In our case we used Endnote[1] to record the search results. After source selection identification, we identified the search string based on keywords, their synonyms and alternative words related to debugging of concurrent and multicore software.

It is important to note that concurrent and multicore software are also directly related to parallel, many-core and multi-threaded software. Thus we decided to consider such keywords as part of the search string. We also added additional terms, i.e., debugging, troubleshooting, error detection and testing, which also could contain information about concurrent and multicore bugs and the debugging processes. Furthermore, the terms software and system were included to restrict the scope of search. Related papers published between 2005 and 2014 were included in our pool in order to investigate the most recent relevant papers and understand the maturity and weight of the research performed.

We used the Boolean operator OR to joined synonyms and keywords and an AND operator for major terms. Thus the complete search string includes three parts and is formulated as follows:

```
(concurrent OR multicore OR parallel OR multi threaded OR multi-threaded OR pseudo parallel OR multi-core OR many-core ) AND (software OR system) AND (debugging OR error detection OR troubleshooting OR testing)
```

[1] http://endnote.com.

After determining the source selection and defining the search string, we performed a search for available papers in order to find the available references matching our criteria.

2.3 Study selection criteria (Step 3)

The study selection criteria (Step 3) involves selecting relevant studies from the search results. This step is one of the most time-consuming and difficult parts in a systematic literature review (Keele 2007). Fig. 2 shows the different stages in the study selection process.

After completing the last step (Step 2), we had 7776 papers in our Endnote database. Out of these, 1005 were duplicates that we removed. Then out of the remaining 6771 papers, we excluded a total of 598 books. The remaining 6173 papers were subjected to a multi-step inclusion/exclusion process. Since the focus of this study is on software and particularly debugging of concurrent and multicore software, we decided to look for the keywords that could help us remove a bulk of irrelevant papers. We call this step Bulk Removal. These keywords helped us exclude studies that were not compatible with the scope of this mapping study (e.g., keywords related to hardware aspects and testing of specific hardware such as system-on-chip, FPGA, voltage and CMOS). The total number of papers excluded by applying bulk removal was 1325.

The remaining papers were grouped into three classes: Relevant (R), Non-Relevant (NR) and Not-Clear (NC) based on reviewing the titles and abstracts (Title + Abstract Exclusion). If a researcher could not recognize the relevance of a paper after reading its title and abstract, the paper was classified as a NC paper.

Specifically, each researcher marked a paper as R if it satisfied two fundamental questions. (1) Does the paper discuss any type of concurrency bug(s)? (2) Does the paper focus on any stage of the debugging process? These questions were defined based on the research goals and research scope. If a paper mentioned any type of concurrency bug (as explained in Sect. 3.1), the researcher would answer "Yes" to the first question. If the paper discussed or focused on any stage of the debugging process (as explained in Sect. 3.2), the researcher would answer "Yes" to the second question. If the answer was "Yes" for both questions, then the paper was marked as R. In case at least one question was answered with "No", the paper was marked as NR. In case the researcher was uncertain, the paper was marked as NC.

In order to increase the reliability of our study selection, we applied a systematic voting process (group majority vote exclusion) among the team members for NC papers. Our voting mechanism was based on the two mentioned questions. Each researcher independently answered these two questions for each NC paper. If a given paper received at least three R answers (from four voting researchers), then we considered it as R and a paper with three NR votes was classified as NR, while the remaining papers were classified as NC papers. However we noticed that after the *group majority vote exclusion* sub-process, 43 papers were still NC for all researchers. Thus we decided to do a full-text skimming of these NC papers. As a result, all papers could be classified as either R or NR.

As shown in Fig. 2, the initial number of collected papers were 7776 papers. After applying the explained sub-processes, the number of paper decreased to 145.

To decrease the risk of missing important and relevant references, we added a secondary search step. We browsed the volumes of *International Journal of Parallel Programming*[2] between 2013 and 2014. We also browsed the proceedings of *International Workshop on*

[2] http://www.springer.com/computer/theoretical+computer+science/journal/10766.

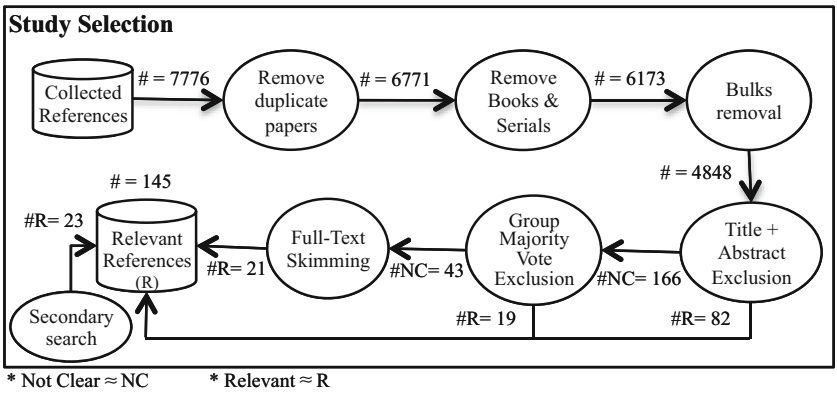


Fig. 2 Study selection process

Program Debugging (IWPD)[3] from 2010 to 2014. These two venues were selected because they had published primary studies relevant to our research area. We read the titles and abstracts of papers, compared each paper with our study scope and if they were relevant and not already in our database, we added them as a R papers.

Based on the knowledge gained in the initial search, as well as additional input from experts in the field, a secondary search with more specific terms was undertaken. This can be viewed as a validation of the work in the initial search, adding confidence in the initial 122 papers selected by mitigating the validity threat of missing papers. In this secondary search, the following search string was used:

(synchronization OR synchronous OR atomic OR mutex OR asynchronous OR CCS OR Pi-Calculus OR deadlock OR dead-lock OR live-lock OR live-lock OR starvation OR suspension OR race OR "order violation" OR order-violation OR "atomicity violation" OR atomicity-violation) AND (debugging OR "bug identification" OR "cause identification" OR "exploring correction" OR correct OR correcting OR fix OR fixing OR "bug localization" OR "bug detection" OR "root cause")

This secondary search also included applying backward snowballing (Jalali and Wohlin 2012) using a subset of the originally identified primary studies. Particularly, we reviewed the reference lists of these papers to identify missing relevant papers to include. The secondary search initially yielded 79 papers that underwent the same exclusion steps as the papers found in the original search. As shown in Fig. 2, the secondary search identified 23 additional relevant papers. This brings our total to 145 papers relevant according to our criteria.

2.4 Data mapping (Step 4)

The data mapping is based on the following four attributes: *debugging process*, *type of concurrent bug*, *research type* and *contribution type*. Complete explanations for each of these are provided in Sect. 3. As presented in Fig. 1, this step was performed after

[3] http://paris.utdallas.edu/iwpd14/index.html.

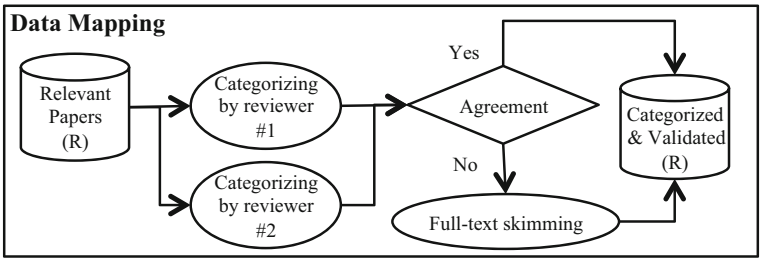


Fig. 3 Mapping process

applying the study selection process, i.e. only the final set of papers was categorized. Fig. 3 illustrates the data mapping process. Each paper was reviewed by at least two researchers. Each reviewer proceeded as follows: In case the focus (i.e., bug identification) was explicitly mentioned by the authors, we tagged the paper accordingly. Otherwise, the paper was studied more in-depth for finding a correct classification.

If both reviewers' classification for each paper were the same, the categorization was agreed upon. Otherwise we did full-text skimming in order to categorize the paper(s). During the data mapping step, we found that some papers could be classified into more than one category according to our classification schema. Thus we classified them into all relevant categories. Finally, the output of this step was a set of papers that were categorized and validated in identified classes.

3 Study classification schemes

In addition to general publication data (i.e., year, venue, author and affiliation), we extracted four main facets of data for the mapping process: *debugging process*, *type of concurrency bug*, *research type* and *contribution type*. *contribution type* are based on the classification process described by Petersen et al. (Petersen et al. 2008) (Sect. 3.3). For *Debugging process* and *type of bug*, we used the classification scheme for concurrency bugs described in (Abbaspour et al. 2015) (Sects. 3.1, 3.2). For *research type*, we used the scheme proposed by Wieringa et al. (Wieringa et al. 2005) (Sect. 3.4). These four classification schemes are described in detail below.

3.1 Debugging process classification

From an industrial perspective, a simple life cycle of a software problem defined by (Zeller 2009) is: (1) User reports a problem to software provider. (2) A developer from software provider reproduces the problem. (3) The developer isolates the circumstances of the problem. (4) The developer fixes the problem locally. (5) The developer delivers the fix(es) to the user. Typically based on the management strategy of an organization, other processes can be added to this life cycle. For instance, one step within large organizations, is called *bug triaging* (Anvik et al. 2006), involving the routing of the bug handling to the right part of the organization or to the right team involved in developing the part of the software found buggy. This process is more important in an agile development context, where many teams might be involved in developing the same part of the code. To be able to determine the problem, often the bug is replicated and information gathered.

Determining the set of steps to be able to repeat and gather information is frequently called *troubleshooting*. Troubleshooting is often performed by testers if the reason was a failed test case, but could also be by e.g., support personnel or developers. At some point the bug will reach a developer and often here the actual debugging process starts. The next step is identifying the right part of a software component, typically a smaller part of the software, e.g., an identity like file or files that are involved in the software. This phase is frequently called *bug or fault localization*. Then, the bugs and their location must be found (Schneider 2014) before the root cause can be identified. At this point we assume that the developer have at least pinpointed the files, code sections and general location of the bug, by utilizing e.g., minimization techniques (Zeller 2009) and been able to reproduce the bug in context.

It should be noted that the debugging process starts at different times in different types of organizations and teams. In a small team with few developers, it is normally clear what part of the code is in question when a program executes unsuccessfully or a test case fails. Here, typically, the developer has to find the bug (Wang et al. 2014). In larger organizations, usually the first sign of any bug is the failure of the software or system. The bug fixing process then starts with the submission of an anomaly report.

Below, we provide a staged classification of the different phases in the debugging process, derived from the primary studies included in this review. This classification discusses the stages that follow after a software failure has been observed, and its root cause should be determined and corrected. We define the debugging process to consist of the following five stages:

1. *Bug identification* is the process of finding the approximate location of a bug (in terms of source code unit, sub-system or even organizational unit), such that the remainder of the debugging process can be assigned to the appropriate stakeholder. It is to be noted that the scope of our definition of bug identification covers terms such as bug localization and bug detection. In case the failure was detected during testing, bug identification is usually performed by the testing team and followed by a team review to prioritize fixes (Zaineb and Manarvi 2011).
2. *Type of bug identification* is a process to help developers in finding the real cause of bug by understanding the type of bug. In (Abbaspour et al. 2015), we extended the common debugging process by adding a sub-process that suggests that before type of bug identification developers could check the properties of identified bug(s) and compare them with the properties given for each class of concurrency bugs. Thus, developer(s) can recognize the potential type of bug.
3. In *cause identification*, the root cause of a bug is identified. Since the root cause refers to the most basic reason(s) for the occurrence of a bug, during this process a bug can reasonably be identified by a developer or the debugger (e.g., variable A was the root cause of bug for variable B or lock (1) was the root cause of bug number 5).
4. The process of *exploring corrections* can be applicable when we have more than one possible solution for fixing the bug. Typically the potential solutions are compared and the best solution for the current bug is selected.
5. Finally *fixing bug* is the process for repairing and fixing the current bugs. It is the last stage of the debugging process in order to remove the bug.

3.2 Concurrency bug classification

One of our main contributions is a better understanding of the different types of concurrency bugs. We classified and mapped the relevant papers related to the types of

concurrency bugs using a classification of concurrency bug types based on observable properties (Abbaspour et al. 2015). This classification categorizes concurrency bugs into seven disjoint classes. The bug types are listed below and are also presented in summarized form in Table 1. The properties in this table are used to distinguish different types of concurrency bugs and that it is assumed that a concurrency bug has occurred, i.e., the properties may not be sufficient to identify a bug.

As listed in Table 1, the categories of that classification are as follows:

- *Deadlock* happens when a thread (process) cannot proceed because it needs to obtain a resource which is held by another thread, while it holds a resource that the other thread needs. During deadlock, all involved threads are in a waiting state.
- *Livelock* happens when a thread is waiting for a resource that will never become available, while the CPU is busy releasing and acquiring the shared resource. It is similar to deadlock except that the state of the process involved in the livelock constantly changes and is frequently executing without making progress.
- *Starvation* happens when a process is indefinitely delayed because other processes are always given preference. During starvation, at least one of the involved threads remains in the ready queue.
- *Suspension* occurs when a thread (that is not deadlocked) waits for an unacceptably long time in a queue to acquire a lock for accessing a shared resource.
- *Data race* occurs when at least two threads access the same memory location, with at least one of them writing data to that location and these accesses are not protected by locks.
- *Order violation* occurs when the expected order of at least two memory accesses is not respected, i.e., the programmer's intended order of execution is not enforced.
- *Atomicity violation* happens when the execution of two code blocks (sequences of statements protected by lock) in one thread is concurrently overlapping with the execution of one or more code blocks of other threads, in such a way that the resulting content of memory cannot be achieved by executing the involved blocks in any non-overlapping order.

3.3 Type of research contribution classification

The *type of research contribution* facet helps us to observe trends in the community's research focus, such as increased focus on developing new debugging techniques or new debugging tools.

In order to categorize the included studies based on research contribution, we use a classification scheme similar to that defined by (Petersen et al. 2008) and (Engström and Runeson 2011). In our case, this classification is derived from keywords and includes five classes:

- The *methods* (techniques/approaches) category includes papers describing how to perform debugging of concurrent and multicore software. We include papers with methods describing general concepts but also papers with more specific and detailed working procedures.
- The *models* (frameworks) category is focused on representations of information to be used to support the actual debugging. Other examples of papers in this class are models that aim to prevent specific concurrent and multicore bugs.

Table 1 Concurrent software bugs properties

Property	Deadlock	Livelock	Starvation	Suspension	Data race	Order violation	Atomicity violation
At least one thread t ∈ T_b is in the Waiting state	✓			✓		✓	✓
At least one thread t ∈ T_b is the Executing state		✓	✓	✓	✓	✓	✓
At least one thread t ∈ T_b is in the Ready state			✓				
All threads in T_b have read and written to a spinlock variable		✓					
All threads in T_b are waiting for a lock held by another involved thread	✓						
At least one thread t ∈ T_b is in the ready queue for an unacceptably long time			✓				
At least one thread t ∈ T_b is in Waiting state for an unacceptably long time	✓			✓			
All threads in T_b are in Executing state					✓		
No thread t ∈ T_b is able to proceed and progress	✓	✓					
There are incorrect or unexpected results					✓	✓	✓
The number of threads in T_b is larger than the number of free processor cores			✓				
Potential request to a resource is larger than the number of available resources of that type				✓			
All threads in T_b hold a lock	✓						
At least one of the threads t ∈ T_b holds a lock	✓			✓		✓	✓
Accesses to shared memory were made from different threads in T_b					✓	✓	✓

Table 1 continued

Property	Deadlock	Livelock	Starvation	Suspension	Data race	Order violation	Atomicity violation
At least one of the memory accesses was Write					✓	✓	✓
Accesses to shared memory were targeted the same memory location					✓	✓	✓
The memory accesses were NOT protected by a synchronization mechanism					✓		
There was at least one correct execution ordering between the memory accesses which the program failed to enforce						✓	
An atomic execution of statements was required							✓

In the property list, when they refer to **threads t**, they are referring to the threads in the set $T_b \subseteq T$ when T_b refers to the set of threads directly related to the bug

- The third class is *metrics*, which provide new or specific measurements and measure certain properties in debugging phase. An example of a measurement in this category is measuring the speedup of bug detection algorithms or measuring the advantages or disadvantage of an available method.
- *Tools* is our fourth category which refers to any kind of tool or tool support for concurrent and multicore software debugging. In the Tools category we typically find research prototypes.
- Our final category *open items* includes the remaining papers that include issues not covered by the other categories above.

3.4 Classification of research types

The justification for using *research type* as a dimension in the classification is to aid us to understand the maturity and weight of the research performed and in that perspective better understands the research approach in general, independently of a specific focus area For research type, we use the classification scheme described by (Wieringa et al. 2005) without changes. This taxonomy reflects the research approaches used in the papers. It is a general taxonomy, independent from any specific focus area of research, and includes six classes:

- *Validation research* concentrates on investigating a proposed solution, which is novel and has not yet been implemented in practice. Investigations are carried out systematically, i.e., prototyping, simulation, experiments, mathematical systematic analysis and mathematical proof of properties.

- *Evaluation research* focuses on evaluating a problem or an implemented solution in practice, i.e., case studies, field studies and field experiments.
- *Solution proposal* is a novel solution for a problem or new significant extension to an existing technique.
- *Conceptual proposal* describes a new way of looking at things by structuring in form of a conceptual framework or taxonomy.
- *Opinion paper* expresses the authors opinion whether a certain technique is good or bad
- *Experience paper* sketches on personal experience of the author, i.e., what and how something has been done in practice.

4 Concurrent and multicore software debugging: a map of the field

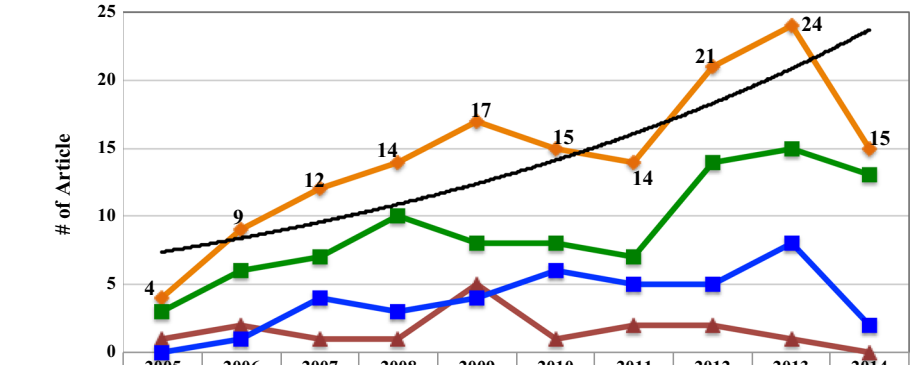
The search process identified 145 primary studies on concurrent and multicore software debugging published between 2005 and 2014. The full list of identified primary studies is provided in Table 4. In this section we present our synthesis of the primary studies identified in the field. All the raw data are available at https://goo.gl/krI01J.

4.1 Publication trends between 2005 and 2014

By analyzing the distribution of included primary studies, based on year of the publication, publication venue, institution and authors, we get an overview of the size and evolution of the field in the last decade.

4.1.1 Distribution of publications

Figure 4 shows the number of papers published on concurrent and multicore software debugging per year from 2005 to 2014. In our investigation we observed that, in 2013 there



	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014
Total	4	9	12	14	17	15	14	21	24	15
Academia	3	6	7	10	8	8	7	14	15	13
Industry	1	2	1	1	5	1	2	2	1	0
Both	0	1	4	3	4	6	5	5	8	2

Fig. 4 Distribution of primary study by publication year

was a merge between one of the most relevant workshops in the area (PADTAD[4]) and another conference, not fully focused on the topic, resulting in MUSEPAT.[5] This may have impacted the result for 2014. This new conference was canceled in 2015, and the relevant papers seem to have been moved to SAC 2015.[6] We will further discuss the distribution in Sect. 4.1.2.

As shown in Fig. 4, the trend (the black line) shows an increased attention in recent years, with the number of publications increasing from 4 in 2005 to 24 in 2013 (six times more, though not being a continuous increase).

In general, our interpretation of Fig. 4 leads to two observations. First, developers (and other roles performing debugging) have refined the field and are thus better in categorizing and uniquely identify new types of concurrency bugs. One example is the previous category described as data race condition. The interpretation has evolved into improving the definitions and identification taking new aspects of time and order into consideration, resulting in new categories, e.g., atomicity violation. As a result, the new bugs will require suitable and improved strategies and mechanisms during the debugging process. Second, there are still unsolved problems in debugging concurrency bugs, which leads researchers to spend more effort in this area and propose new solution(s) for current unsolved issues. Thus there is a need to improve and refine the current knowledge by providing new solutions and evaluating them in real-world software.

4.1.2 Main publication venues

In Table 2, the most frequent publication venues are ranked for the included primary studies. We found that the included set of primary studies was published through a total of 95 conferences and journals. Table 2 lists the publication venues that had three or more papers. The list represents about 30 % of the total number of studies.

Around 10 % of the published papers belong to the ACM SIGPLAN Notices journal. the second rank belongs to workshop on Parallel and Distributed Systems: Testing, Analysis and debugging (PADTAD). PADTAD was thus the main workshop for this area, with eight published papers. Due to the reorganization of venues described in Sect. 4.1.1, ACM/SIGAPP Symposium On Applied Computing (SAC) will probably be the main venue in the future. Third ranked is International Symposium on Software Testing and Analysis (ISSTA) with five published papers. ISCA and IPDPS are the forth ranked in the list with four published papers each, followed by PLDI, ASE and ICSE with three papers each.

The summarized result in Table 2 can help new researchers find the most relevant publication venue(s) and articles.

For the remaining venues, two or less papers have published. This means that 70 % of published papers are scattered over various venues. The multitude of venues for publication shows that the community does not have an obvious targeted place of publishing. Thus could indicate that there is a need for the community to organize more workshops and conferences targeting concurrent and multicore software debugging.

[4] http://faculty.uoit.ca/bradbury/padtad2012/.

[5] http://eventos.fct.unl.pt/musepat2013/.

[6] http://faculty.uoit.ca/bradbury/sac-musepat2015/.

Table 2 Top five publication venues

Rank	Publication channel	Venue	#	%
1	Journal Article	ACM SIGPLAN Notices	14	9.66 %
2	Conference Proceedings	Workshop on Parallel and Distributed Systems: Testing, Analysis, and Debugging (PADTAD)	8	5.52 %
3	Conference Proceedings	International Symposium on Software Testing and Analysis (ISSTA)	5	3.45 %
4	Conference Proceedings	International Symposium on Computer Architecture (ISCA)	4	2.76 %
4	Conference Proceedings	International Symposium on Parallel and Distributed Processing (IPDPS)	4	2.76 %
5	Conference Proceedings	ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)	3	2.07 %
5	Conference Proceedings	International Conference on Automated Software Engineering (ASE)	3	2.07 %
5	Conference Proceedings	International Conference on Software Engineering (ICSE)	3	2.07 %

4.1.3 Academia and industry representation

Figure 5 provides a view of the affiliations of the authors of the primary studies: A paper with only academic authors is categorized as *academic*, a paper with only authors active in industry is categorized as *industry*. If the paper is written by both academic and industrial authors, the paper is categorized as *both*. Not surprisingly, academia shows a greater engagement in publishing results about debugging of concurrent and multicore software, with a share of two thirds of all published papers, and only 11 % papers have authors exclusively active in industry. In addition, almost one forth (26 %) of published papers are joint papers written by authors from both academia and industry.

Clearly, as for most scientific publication, academia is dominating in the field, although the number of papers authored (or co-authored) by industry is as high as 37 %. Although we are not able to draw any firm conclusions, this could be an indication of substantial industrial interest in the research, or simply be a consequence of debugging being a regular task in industry.

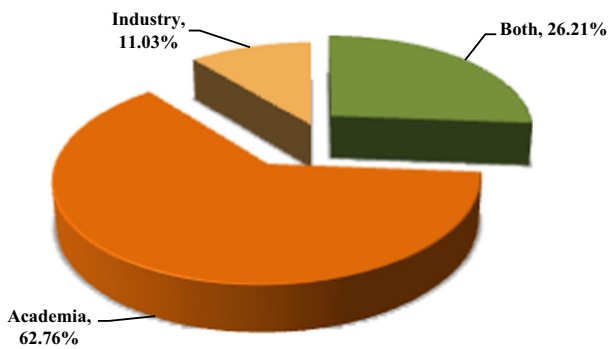


Fig. 5 Distribution of Academic, industry and joint publications

However, since debugging is one of the tasks in the software development life-cycle practice it seems reasonable to claim that more reliable evidence could be obtained by more frequent consideration of real data from real-world software. This claim is, however, not firmly supported by our study, as academic results could be based on industrial data without including industrially affiliated co-authors. We believe there is a need for industry researchers to contribute more in publishing new research and sharing their idea(s) and solution(s) in this field.

4.1.4 Active research organizations

By analyzing the affiliation of the authors for the included studies, we found that 131 unique institutions participated in the publications. University of California has the largest contribution with 9 % of the total number of papers. Among the industrial affiliations, IBM Research had the largest number of papers when we combined data from their three different contributing sites. The most active institutions in the field are presented in Table 3. An interesting note is that the companies IBM, NEC and Microsoft are active in the area by publishing externally. University of California leads the production activity, with 13 papers over University of Illinois (11) and then Gyeongsang National University (8).

Table 3 can guide new researchers and interested companies to find relevant research centers and institutes to collaborate with. Further, by clarifying the main drivers, we provide basis for collaboration to arrange common workshops and conferences, and hopefully contribute to reducing the current scattered publication situation.

4.2 Focus and potential gaps in existing work

One of the key objectives of this study is to provide an overview of the primary issues and problems in concurrent and multicore software debugging and to identify solutions that have been proposed to address these. For this purpose, we have extracted and analyzed data concerning specific *debug process*, *concurrency bug type*, *research contribution* and *research type* of all identified studies.

4.2.1 Concurrency bug focus

We analyzed the distribution of papers based on our classification of different types of concurrency bugs, described in (Abbaspour et al. 2015). As described in Sect. 3.2, this

Table 3 Top five most productive and active research institutions

Rank	Institution	# of Articles
1	University of California, USA	13
2	University of Illinois, USA	11
3	Gyeongsang National University, South Korea	8
3	IBM Research Center, USA, Japan and Israel	8
4	Microsoft Research Center, USA, UK and Spain	7
5	NEC Laboratories America, USA	6
5	University of Michigan, USA	6

taxonomy holds seven types of concurrency bugs: (1) Deadlock, (2) Livelock, (3) Starvation, (4) Suspension, (5) Data race, (6) Order violation and (7) Atomicity violation. For the sake of this review, we have added two more categories to the taxonomy: (8) *General* and (9) *Others*.

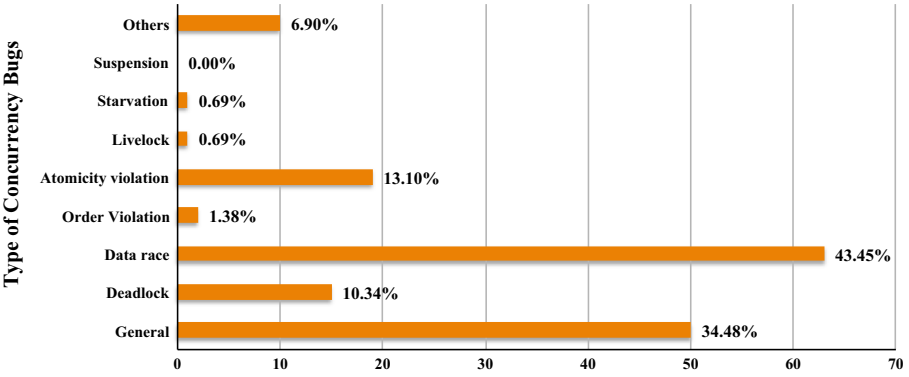
The *General* category includes papers that address concurrency bugs in general, without focusing on any specific type. The *Others* category includes papers that cover bugs related to concurrency and parallelism, but that do not lend themselves to classification according to the defined taxonomy. Examples of such bugs include non-deadlock and shared memory bugs.

Figure 6 shows the concurrency bug categories and how frequently they are investigated, by adding the number of papers per category. All 145 papers were taken into account.

In these 145 papers, 161 concurrency bugs were addressed (as some papers addressed more than one bug). About one-third (34 %) of the papers focus on concurrency bugs in general. More than 43 % of the publications that discuss a particular debugging aspect of a bug, focus on data races, a well-known and a common concurrent bug (Qi et al. 2012). The focus on data races also holds for the top five contributing institutions shown in Fig. 3. For instance, data races are the primary focus of the papers published by the University of California. Similarly, from the industrial perspective, data race is the most researched bug type in the publications by *IBM Research Center*.

On the other hand, none of the reviewed papers address issues related to suspension-based locking. We believe there could be two reasons for this: First, suspension might not be a common terminology and second, this bug might not represent a common problem.

In general, our interpretation is that there is a need to propose new studies in debugging for concurrency bugs such as suspension, starvation, order violation and livelock. The result indicates that there are few papers addressing these type of bugs.



	General	Deadlock	Data race	Order Violation	Atomicity violation	Livelock	Starvation	Suspension	Others
# of addressed bugs	50	15	63	2	19	1	1	0	10

Fig. 6 Concurrency bugs distribution

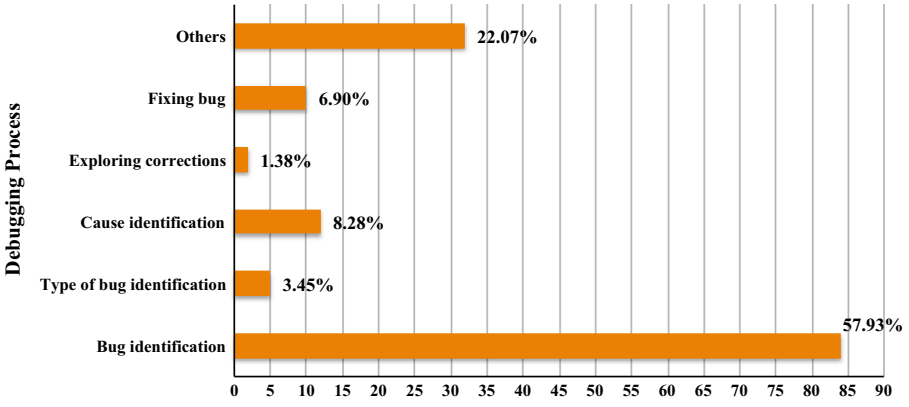
4.2.2 Debugging process phase focus

To better understand where in the debugging process research is focused, we have made a classification. In addition to the debugging process described in (Abbaspour et al. 2015) and Sect. 3.1, we have added one more class (*others*) in order to accommodate studies that are related to debugging of concurrent and multicore software but that do not trivially relate to any part of the debugging process described in Sect. 3.1. The focus of such studies include, e.g., protection from bugs, pre-debugging, conflict checking, bug prevention and reviewing of debugging methods. Thus, type of debugging process is classified into six categories: (1) bug identification, (2) type of bug identification, (3) cause identification, (4) exploring corrections, (5) fixing bug and (6) others. The first five of these categories are described in Sect. 3.1.

Figure 7 presents the frequency of contributions focusing on different parts of the debugging process. More than half of the primary studies (76 of 145, 58 %) focus on *bug identification*, while just 2 primary studies (∼2 %) have a focus on *exploring corrections*.

Figure 7 also shows that ∼7 % of papers focus on later stages of the debugging process (*exploring corrections* and *fixing bugs*), while a large majority of papers focus on earlier phases of the debugging process, primarily *bug identification*.

There is a shortage of papers related to *exploring corrections* and *fixing bugs* in the final stage of the debugging process. This is probably due to the similarities of fixing a concurrent bug compared to fixing any other bug and can thus be viewed as a solved problem, or at least not a problem in need of further research. It seems clear that the main caveat in the debugging process is to identify and recognize (as well as reproduce) complex concurrency bugs. Once targeted, the corrections are understood and could be considered relatively straight forward. Instead the focus here should be to better evaluate and compare different solutions in terms of design (correction) patterns for concurrency bugs.



	Bug identification	Type of bug identification	Cause identification	Exploring corrections	Fixing bug	Others
#	84	5	12	2	10	32

Fig. 7 Debugging process distribution

4.2.3 *Relation between research type, research contribution and type of concurrency bugs*

Figure 8 shows a map over concurrency bug types, distributed over types of research (vertical axis) and types of research contributions (horizontal axis). Each pie chart represents the number of publications mapped to this specific combination, and different colors show the different types of concurrency bugs within that combination. With respect to research type, *method* and *tools* are prominent categories. With respect to research contribution, *solution proposal* is a dominant category.

A majority of the primary studies, 81 of 145 (∼56 %) provide a *solution proposal* with a specific type of *method*. The key bug type investigated and provided solutions for is data race (43 %). Large research efforts are concentrated on *tools*. Clearly, debugging specific problems are tool intensive, so this is an obvious aspect, where one also finds *solution proposal*, with 31 published papers (∼21 %). In this case too, data race is the most prominent bug type.

Figure 8 also shows that the majority of the primary studies based on *contribution type*, *type of concurrency bug* belong to *method* type by about 66 % of the total number of published paper. On the other hand, the majority of primary studies in *research type* based on *type of concurrency bug* belong to *solution proposal* with 90 % of total published papers.

In summary, based on the results from Fig. 8, we can interpret that there are many gaps in each class. These gaps mostly belong to *experience papers* in all types of research contributions (just 2 studies out of 145). However, *conceptual proposals* are less attractive for researchers targeting aspects of the model, metric and tools categories. What is more

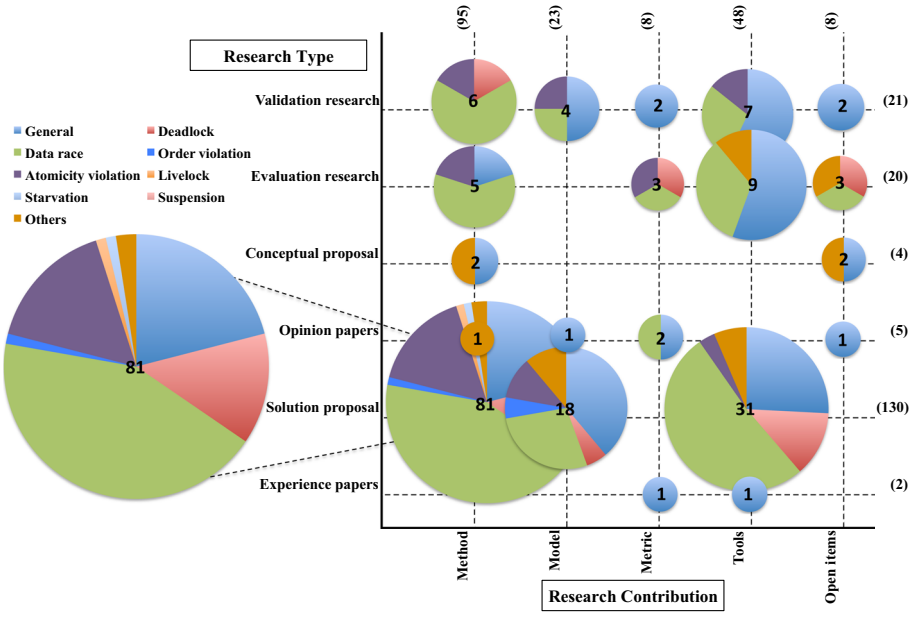


Fig. 8 Relation between research contribution, research type, type of concurrency bugs and number of papers

concerning is that there are 10 classes with just one or two published papers and 10 classes without any publications. Possible explanations are that many aspects of this area are either already solved or the researchers in this field have not realized the potential problems. We also believe that there is a specific need to have industrial researchers shares their experience by publishing *experience papers* in this area.

In addition, by comparing the total papers in the *solution proposal* class and *evaluation research* class, it is obvious that there is a gap in the *evaluation research* class. Thus we could conclude that there is a need for further research of this type in order to implement valid solution(s) and evaluate problems in practice.

Since tools can play important role in debugging, Fig. 9 presents the distribution of relevant papers, which propose a solution for concurrency-debugging tool(s) considering the concurrency bug type. As shown in Fig. 9, we identify 31 of 145 primary studies ($\sim$21 %) that proposed a new solution by a tool. The largest class of concurrency bug is here data race with 52 % of primary papers in this category.

From the industry contribution point of view, we analyzed the industry involved distribution publications. Based on our investigation, for all types of research contribution we obtained the following results: The industry involvement in *method* class is 28 % of total published paper, in *model* class $\sim$5 %, in *metric* class $\sim$3 %, in *tools* class $\sim$16 % and in *open items* $\sim$6 %. Given that industry involvement is present in about one-third of the primary studies, it is noteworthy that only 3, 5 and 6 % of the bugs addressed by industry papers are in the form of *method*, *metric* and *open items* papers, respectively.

In general, we can conclude a lack of research proposing new solution(s) as tool for specific types of bugs such as *order violation*, *atomicity violation*, *suspension* and *starvation*. We believe that if industrial and academic researchers could focus their attention more on these gaps, then there is a good chance that future tools could simplify the debugging process for concurrent and parallel applications. Consequently, it would help to increase software quality. In other words, there is a need for all type of concurrency-debugging tools to address multiple types of concurrency bugs.

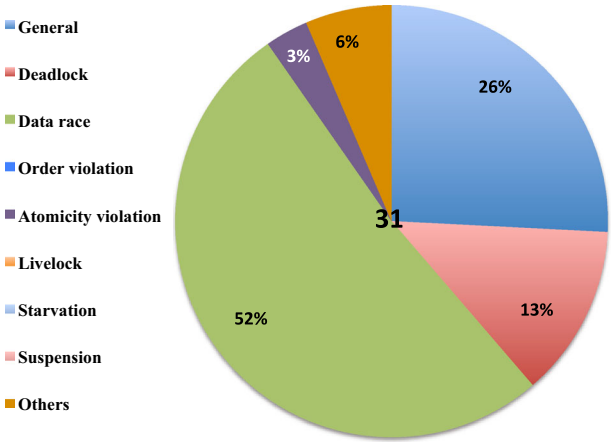


Fig. 9 Distribution of concurrent bug for *tools* and *solution proposal* category

4.2.4 Relation between research contribution, research type and debugging process

In Fig. 10 we look at the mapping of contribution type (horizontal axis) and the research type (vertical axis) but instead of bugs, we look at the different phases of the debugging process. Each bubble represents the number of primary studies, and the different colors represent different phases in the debugging process. It is important to note that the vast majority of the primary studies (66 of 145, 45 %) focus on the bug identification phase, proposing a solution by providing a method. This means that identifying (or recognizing) bugs is clearly the main issue addressed in published research, and the second most common issue is providing tools. It is interesting to note that in almost all cases but modeling, which in itself is a "prevention" of a specific bug—this is the main issue and also difficult to solve.

The identified study gaps based on research type, contribution type and debugging process are presented in Fig. 10.

Based on our investigation, the total number of primary studies that proposed a method are 79 (∼54 %), whereas the number of studies that proposed a model is 22 (∼15 %). There are 46 (∼32 %) papers describing a tool and 8 (∼5 %) studies that describe the use of metrics. Open items are discussed by 5 (∼3 %) studies.

As shown in Fig. 11, we identified 30 of 145 primary studies (21 %) that propose a new solution as a tool. The largest class of debugging process belongs to bug identification by 50 % of primary papers in this category. However, as shown in Fig. 11 just 7 % of primary studies in this category propose a new tool for fixing concurrency bugs.

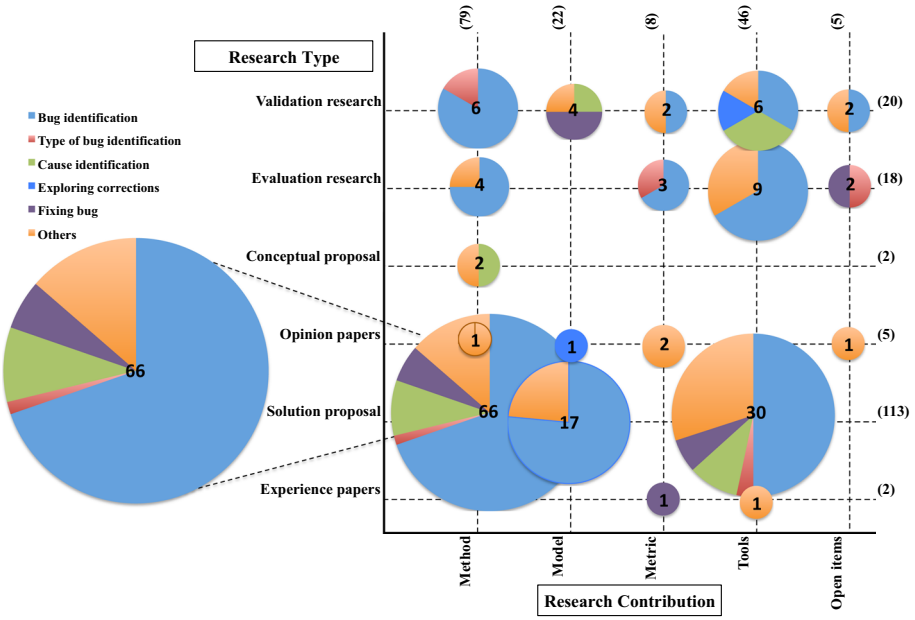


Fig. 10 Relation between research contribution, research type and type of debugging process and number of papers

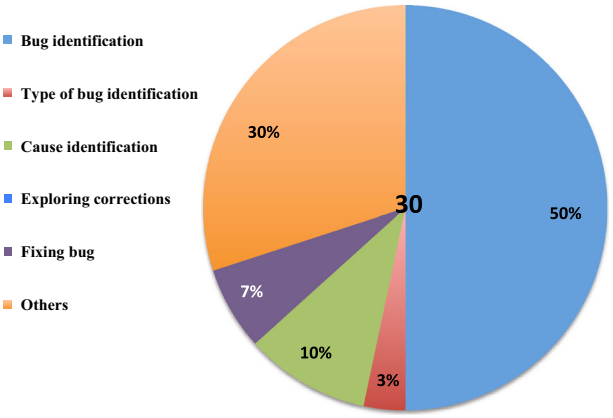


Fig. 11 Distribution of debugging process for *tools* and *solution proposal* category

Exploring tools and tool support in order to address fixing the bugs (or finding ways to prevent them), we can interpret the result as a significant need to put more effort on *experience* and *evaluation* studies.

4.2.5 Development of research relating bug type and debugging process

The left-hand side of Fig. 12 presents the amount of primary studies focusing on a particular type of concurrency bug, and a particular phase in the debugging process. The right-hand side of the figure illustrates the year-wise distribution of primary studies per concurrency bug type. The bubble at each intersection of axes contains the number of primary studies on the topic.

The recent research trend for concurrency bug type does not show much difference from earlier trends. Some possible conclusion could be: (1) debugging concurrent programs is very hard; (2) debuggers are not motivated enough to deal with concurrency bugs; (3) debuggers are not familiar with different types of concurrency bugs; (4) there are not

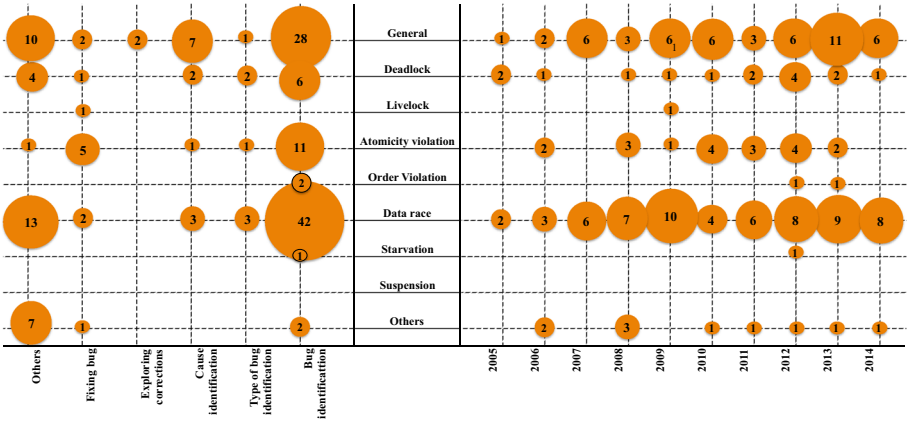


Fig. 12 Relation between *type of concurrency bug*, *type of debugging process* and *publications year*

Table 4 List of included primary studies, categorized by debugging process [**DP**, i.e., bug identification (BI), type of bug identification (TBI), cause identification (CI), exploring corrections (EC), fixing bug (FB), others (O)], and bug type [**BT**, i.e., general (G), livelock (L), deadlock (D), atomicity violation (AV), order violation (OV), data race (DR), starvation (S), others (O)]

DP/BT	References
BI/G	Wu et al. (2012), Oßner and Böhm (2013), Uhrig (2011), Wang et al. (2007), Tzoref et al. (2007), Montesinos et al. (2008), Trainin et al. (2009), Fonseca et al. (2011), Chen and MacDonald (2007), Fonseca et al. (2010), Chen and Wang (2009), Andersen (2007), Arulraj et al. (2013), Arulraj et al. (2013), Ball et al. (2009), Vo and Gopalakrishnan (2010), Lu et al. (2007), Wang et al. (2014), Adalid et al. (2014), Bond et al. (2013), Eichinger et al. (2014), Hong and Kim (2013), Mozaffari-Kermani et al. (2014), Rossi et al. (2013), Wang et al. (2014), Zhang et al. (2013), Park (2013), Gesbert et al. (2010)
BI/D	Hilbrich et al. (2012), Wen et al. (2011), Ma et al. (2012), Shousha et al. (2012), Francesca et al. (2011), Shimomura and Ikeda (2013)
BI/AV	Wang and Stoller (2006a), Teixeira et al. (2010), Lucia and Ceze (2009), Wang and Stoller (2006b), Park et al. (2012), Chen et al.(2008), Park and Sen (2008), Flanagan et al. (2008), Tan et al. (2013), Lucia et al. (2010), Zyulkyarov et al. (2010)
BI/OV	Park et al. (2012), Huang et al. (2013)
BI/DR	Sack et al. (2006), Prvulovic (2006), Qi et al. (2009), Park and Jun (2012), Kahlon et al. (2007), Lucia and Ceze (2009), Said et al. (2011), Wang and Stoller (2006b), Negishi et al. (2012), Serebryany and Iskhodzhanov (2009), Wen et al. (2012), Torrellas et al. (2009), Chen et al. (2008), Zhou et al. (2007), Chiu et al. (2011), Eichinger et al. (2010), Park et al. (2007), Ha et al. (2012), Jannesari and Tichy (2008), Sen (2008), Gupta et al. (2009), Kistler and Brokenshire (2011), Ma et al. (2012), Flanagan et al. (2008), Shousha et al. (2009), Yoshiura and Wei (2014), Tan et al. (2013), Huang et al. (2014), Kasikci et al. (2013), Song and Lee (2014), Huang and Bond (2013), Jannesari and Tichy (2014), Kahlon et al. (2013), Lu et al. (2013), Lu et al. (2014), Maiya et al. (2014), Moiseev et al. (2013), Raychev et al. (2013), Wester et al. (2013), Wu et al. (2013), Kim et al. (2007b), Kang et al. (2014)
BI/S	Shousha et al. (2012)
BI/O	Joshi and Sen (2008), Park and Chung (2008)
TBI/G	Rister et al. (2007)
TBI/D	Desouza et al. (2005), Watson et al. (2009)
TBI/AV	Gao et al. (2011)
TBI/DR	Flanagan and Freund (2010), Gao et al. (2011), Desouza et al. (2005)
CI/G	Weeratunge et al. (2010a), Chen et al. (2012), Copty and Ur (2007), Yuan et al. (2010), Martin et al. (2010), Dantas et al. (2008), (Viennot et al. 2013)
CI/D	Chen (2005), Agarwal and Stoller (2006)
CI/AV	Weeratunge et al. (2010b)
CI/DR	Weeratunge et al. (2010b), Tallam et al. (2008), Al-Shabibi et al. (2007)
EC/G	Schuppan et al. (2005), Nanz et al. (2013)

Table 4 continued

DP/BT	References
FB/G	Berger et al. (2009), Lu et al. (2008)
FB/D	Liu and Zhang (2012)
FB/L	Tian et al. (2009)
FB/AV	Jin et al. (2011), Kahlon (2012), Wang et al. (2012), Liu and Zhang (2012), Buttigieg and Briffa (2011)
FB/DR	Tchamgoue et al. (2010), Tian et al. (2009)
FB/O	Gottbrath (2006)
O/G	Kelly et al. (2009), Mar Gallardo et al. (2006), Elmas et al. (2009), Yu et al. (2012), Park and Sen (2012), Wang et al. (2012b), Machado et al. (2012), Chen and Chen (2013), Li et al. (2014), Makela et al. (2013)
O/D	Gottschlich et al. (2013), Pun et al. (2014), Schaeli and Hersch (2008), Kim and Jun (2010)
O/AV	Gottschlich et al. (2013)
O/DR	Qi et al. (2012), Sadowski and Yi (2009), Wen et al. (2009), Wang et al. (2012a), Veeraraghavan et al. (2011), Schaeli and Hersch (2008), Wang and Fang (2005), Hower and Hill (2008), Tchamgoue et al. (2012), Altekar and Stoica (2009), Pun et al. (2014), Chiu et al. (2011), Kim et al. (2007a)
O/O	Kiefer and Moser (2013), Dinh et al. (2014), Khoshavi et al. (2012), Lonnberg et al. (2011), Devietti et al. (2010), Vasudevan et al. (2008)

sufficient tools to distinguish the different types of bugs separately and (5) debuggers usually use general or common terms such as "concurrent bug" or "race condition" for any type of concurrency bug. However, in compare with early studies there are more studies during last 2 (or 3) years to deal with debugging concurrency bugs.

It is evident from the figure that *bug identification* is the most widely studied process with 92 papers ($\sim$63 %) across different types of bugs. Among this amount, about 45 % of papers focus on data race while no paper was about suspension and livelock. Moreover, very few papers focus on starvation and order violation bugs ($\sim$3 %).

On the other hand, the *exploring corrections* debugging phase has the smallest number of papers across different type of concurrency bugs ($\sim$1 %). The reason for this may be that once the bug can be identified and located, the corrections are often obvious. The 1 % of papers that discuss solutions do not focus on any specific type of concurrency bugs. Table 4 presents the reference list of included primary studies with different combinations. The combinations are the same as presented in the left-hand side of Fig. 12.

5 Threats to the validity of the results

There are several issues that need to be taken into account when conducting a systematic mapping study. These issues can potentially limit the validity of obtained results.

We included papers that were written in English only and there is a chance that we have missed important papers written in other languages. This, we believe, is a limitation with most of the systematic mapping studies. We also decided to include studies that were available only electronically and were published. There is a chance that a relevant paper is not published online due to confidentiality or other reasons. Our mapping study does not extend to such scenarios.

We limited the search for studies within the time span of 2005–2014. This was done for two reasons: (1) to limit the volume of search results for practical reasons; (2) to present more *recent* trends and *current* research gaps in the field. This limitation of years obviously excludes papers published before the year 2005, including highly cited papers such as by (Godefroid 1997). Thus our systematic mapping study is not complete with respect to all research papers on the topic, but instead presents the more recent development in the field.

There is a risk during the study selection process that we excluded a relevant study. However, having more than one researcher reading each paper mitigated this. Further, our additional secondary search with extra keywords in the snowballing fashion added a nice mitigation to this risk. We were fortunate to find additional papers (23), but it is noteworthy to conclude that none of the interpretations changed in any of the charts, except increasing the overall numbers, and one more experience paper was found and made visible in Figs. 8 and 10.

Another threat is related to the classification schema for mapping. Since authors cannot be expected to follow any standard concurrency bug terminology, we partially based our classification of the primary studies on the paper contents using our understanding of its focus. We believe that the process of classification would have been more reliable if consistent terminologies were used in the primary studies. However, multiple iterations among authors should have refined our mapping to mitigate this threat.

The presented classification scheme and obtained results are valid only in our context of computer science and software engineering and does not cover papers from the fields of electronics, mechanical engineering, medical sciences and physics. We tried to have a clear definition of each category within our classification schema. Despite the experience of the researchers, some papers were difficult to categorize due to unclear boundaries between some classification scheme categories, and also due to the way information was presented in those papers.

Finally, as it is known that abstracts do not always reveal the true content of papers, there is a risk that we might have excluded a paper with poor abstract but valid content.

6 Discussion

Based on our investigation, we found that the current taxonomies for concurrent and multicore software debugging properties are lacking coverage of some aspects, specifically aspects of the debugging process. The taxonomies need to include a more complete set of properties with clear explanation for each process step, as well as for sequence of processes steps.

We expected that there would be recent research focused on new types of bugs such as atomicity violation and order violation. However, our investigation indicates that developers, testers, debuggers and researchers still face interesting problems in data race issues. Since this bug is one of the most difficult bugs to reproduce researchers are still challenged by improving the available solutions. Another challenge is the lack of reliable methods to test and debug concurrent and multicore software. This may partly be due to the difficulty to both in industry and academia recognize some types of bugs.

A further observation is that there is no single prominent publication venue, which could be due to the relative novelty of the subject. The multitude of venues for publication shows that the community does not have an obvious targeted place of publishing; thus, the researchers in the subject have not really come together as a community.

We could discuss if researchers from industry should have paid more attention and effort to the final steps of the debugging process, such as exploring correction and fixing the bugs, rather than just focusing on the initial steps of the debugging process (bug identification). It would be natural to assume that industry would target and discuss different solutions and publish evaluations of the best ways to address specific bugs. However, results indicate that the bug identification is the caveat of the process, and once understood and identified, the solutions might be both difficult and involving much work, but is not considered novel enough to be worthy of a research publication. Another explanation might be that industry simply cannot involve too detailed aspects of the software and architecture for public scrutiny involved in the solution, and, as a result, most refrain from publicly announcing particular solutions. We would expect a growth in this area, as the field matures, and as tools and comparative evaluations become more commonplace.

The results of our mapping study also indicate that the current body of knowledge concerning debugging concurrent and multicore software does not report studies on many of the other types of bugs or on the debugging process.

The existing gap(s) on type of bug may be due to the fact that the specific type of bug is not well-known yet, or recognizing them is not an easy task. Moreover, the identified gap(s) on different type of debugging processes could be due to the fact that ether this process is not well defined or not applicable in all software development projects. Also, it could indicate that the process is not easy to apply.

7 Conclusion and future work

This paper provides an overview of existing research on concurrent and multicore software debugging. Based on a structured selection process, we ended up with 145 scientific papers published from 2005 to 2014. By a classification of these studies, we have identified how the scientific research on relevant topics are distributed over the years. The study also helped us to recognize the most common debugging processes used to fix concurrency bugs. Eight detailed research questions were answered by this study. The main results are:

1. The topic has increasingly gained interest since 2005, with the highest number of published papers in 2013.
2. The research was mostly published in conference proceedings ($\sim$66 %) and to a lesser degree in journals ($\sim$34 %).
3. A large variety of conference and journal venues (95) have published the relevant papers in this area.

4. In terms of publication contribution, academia was more active in this area than industry.
5. Authors were from 131 different institutions.
6. Six specific types of concurrency bugs were addressed by articles in last decade.
7. A large fraction of publications addressed data race bugs.
8. Five types of debugging process were addressed by articles in the period, with the bug identification process as the most common one considered.
9. We found six types of research in the selected publications, with solution proposals being the most common type.
10. The published papers essentially focus on four types of contribution, with methods being the most common one.

We could also pinpoint current gaps in research that may represent opportunities for further research on debugging concurrent and multicore software. The more important ones are : (1) *validation* and *evaluation* research in the matter of research type. (2) *Metrics* in terms of research contribution. (3) *Order violation* in terms of concurrency bugs. (4) *Exploring correction* and *fixing bugs* through improved experience reports and evaluations with regard to the debugging process. With this study we have provided an overview of the last 10 years evolvement in the field, identifying gaps in research on debugging concurrent and multicore software.

A general conclusion from our investigations is that even if there are quite a number of results on concurrent and multicore software debugging, there are still quite a number of issues and aspects that have not been sufficiently covered. Considering the industrial importance, there are good grounds to believe that this could be a fruitful area for future research.

Acknowledgments This research is supported by Swedish Foundation for Strategic Research (SSF) via the SYNOPSIS Project.

References

Abbaspour A. S., Hansson, H., Sundmark, D., & Eldh, S. (2015). Towards classification of concurrency bugs based on Observable properties. In Proceedings of the 1st international workshop on complex faults and failures in large software systems. Italy. http://www.es.mdh.se/pdf_publications/3920.pdf.

Adalid, D., Salmern, A., del Mar Gallardo, M., & Merino, P. (2014). Using SPIN for automated debugging of infinite executions of Java programs. *Journal of Systems and Software*, *90*, 61–75.

Agarwal, R., & Stoller, S. D. (2006). Run-time detection of potential deadlocks for programs with locks, semaphores, and condition variables. In Proceedings of the 2006 workshop on parallel and distributed systems: Testing and debugging, pp. 51–60. ACM.

Al-Shabibi, A., Gerlach, S., Hersch, R. D., & Schaeli, B. (2007). A debugger for flow graph based parallel applications. In Proceedings of the 2007 ACM workshop on parallel and distributed systems: Testing and debugging, pp. 14–20. ACM.

Altekar, G., & Stoica, I. (2009). ODR: Output-deterministic replay for multicore debugging. In Proceedings of the ACM SIGOPS 22nd symposium on operating systems principles, pp. 193–206. ACM.

Andersen, D. (2007). Implementing a new application debugging framework for the multi-core age. *Scientific Computing*, *24*(8), 34–35.

Anvik, J., Hiew, L., & Murphy, G. C. (2006). Who should fix this bug? In Proceedings of the 28th international conference on software engineering, pp. 361–370. ACM.

Arulraj, J., Chang, P.-C., Jin, G., & Lu, S. (2013). Production-run software failure diagnosis via hardware performance counters. In ACM SIGARCH Computer Architecture News, Vol. 41, pp. 101–112. ACM.

Ball, T., Burckhardt, S., de Halleux, J., Musuvathi, M., & Qadeer, S. (2009). Deconstructing concurrency heisenbugs. In 31st international conference on software engineering-companion volume, 2009. ICSE-Companion 2009, pp. 403–404. IEEE.

Berger, E. D., Yang, T., Liu, T., & Novark, G. (2009). Grace: Safe multithreaded programming for C/C++. In ACM sigplan notices, Vol. 44, pp. 81–96. ACM.

Bond, M. D., Kulkarni, M., Cao, M., Zhang, M., Fathi Salmi, M., Biswas, S., Sengupta, A., & Huang, J. (2013). Octet: Capturing and controlling cross-thread dependences efficiently. In ACM SIGPLAN notices, Vol. 48, pp. 693–712. ACM.

Brito, M., Felizardo, K. R., Souza, P., & Souza, S. (2010). Concurrent software testing: A systematic review. On testing software and systems: Short papers.

Buttigieg, V., & Briffa, J. A. (2011). Codebook and marker sequence design for synchronization-correcting codes. In 2011 IEEE international symposium on information theory proceedings (ISIT), pp. 1579–1583. IEEE.

Chen, F., Serbanuta, T.-F., & Rosu, G. (2008). jPredictor. In ACM/IEEE 30th international conference on software engineering, 2008. ICSE'08, pp. 221–230. IEEE.

Chen, H. Y. (2005). Analysis of potential deadlock in Java multithreaded object-oriented programs. In 2005 IEEE international conference on systems, man and cybernetics, Vol. 1, pp. 146–150. IEEE.

Chen, J., & MacDonald, S. (2007). Testing concurrent programs using value schedules. In Proceedings of the twenty-second IEEE/ACM international conference on automated software engineering, pp. 313–322. ACM.

Chen, Q., & Wang, L. (2009). An integrated framework for checking concurrency-related programming errors. In Computer software and applications conference, 2009. COMPSAC'09. 33rd annual IEEE international, Vol. 1, pp. 676–679. IEEE.

Chen, S.-Y., Neng, C., Yang, G.-H., Jone, W.-B., & Chen, T.-F. (2012). IMITATOR: A deterministic multicore replay system with refining techniques. In 2012 international symposium on VLSI design, automation, and test (VLSI-DAT), pp. 1–4. IEEE.

Chen, Y., & Chen, H. (2013). Scalable deterministic replay in a parallel full-system emulator. In ACM SIGPLAN notices, Vol. 48, pp. 207–218. ACM.

Chiu, Y.-C., Shieh, C.-K., Huang, T.-C., Liang, T.-Y., & Chu, K.-C. (2011). Data race avoidance and replay scheme for developing and debugging parallel programs on distributed shared memory systems. *Parallel Computing*, *37*(1), 11–25.

Copty, S., & Ur, S. (2007). Toward automatic concurrent debugging via minimal program mutant generation with aspectJ. *Electronic Notes in Theoretical Computer Science*, *174*(9), 151–165.

Dantas, A., Brasileiro, F., & Cirne, W. (2008). Improving automated testing of multi-threaded software. In 2008 1st international conference on software testing, verification, and validation, pp. 521–524. IEEE.

del Mar Gallardo, M., Martnez, J., Merino, P., & Pimentel, E. (2006). On the evolution of reliability methods for critical software. *Journal of Integrated Design and Process Science*, *10*(4), 55–67.

Desouza, J., Kuhn, B., De Supinski, B. R., Samofalov, V., Zheltov, S., & Bratanov, S. (2005). Automated, scalable debugging of MPI programs with Intel Message Checker. In Proceedings of the second international workshop on software engineering for high performance computing system applications, pp. 78–82. ACM.

Devietti, J., Lucia, B., Ceze, L., & Oskin, M. (2010). DMP: Deterministic shared-memory multiprocessing. *IEEE Micro*, *30*(1), 40–49.

Dinh, M. N., Abramson, D., & Jin, C. (2014). Statistical assertion: A more powerful method for debugging scientific applications. *Journal of Computational Science*, *5*(2), 126–134.

Eichinger, F., Pankratius, V., & Bohm, K. (2014). Data mining for defects in multicore applications: An entropy-based call-graph technique. *Concurrency and Computation: Practice and Experience*, *26*(1), 1–20.

Eichinger, F., Pankratius, V., Gro\s se, P. W. L., & Bhm, K. (2010). Localizing defects in multithreaded programs by mining dynamic call graphs. In Testing practice and research techniques, pp. 56–71. Springer.

Elmas, T., Sezgin, A., Tasiran, S., & Qadeer, S. (2009). An annotation assistant for interactive debugging of programs with common synchronization idioms. In Proceedings of the 7th workshop on parallel and distributed systems: Testing, analysis, and debugging, p. 10. ACM.

Engström, E., & Runeson, P. (2011). Software product line testing—A systematic mapping study. *Information Software Technology*, *53*(1), 2–13. doi:10.1016/j.infsof.2010.05.011.

Flanagan, C., & Freund, S. N. (2010). Adversarial memory for detecting destructive races. In ACM sigplan notices, Vol. 45, pp. 244–254. ACM.

Flanagan, C., Freund, S. N., Lifshin, M., & Qadeer, S. (2008). Types for atomicity: Static checking and inference for Java. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, *30*(4), 20.

Fonseca, P., Li, C., & Rodrigues, R. (2011). Finding complex concurrency bugs in large multi-threaded applications. In Proceedings of the sixth conference on computer systems, pp. 215–228. ACM.

Fonseca, P., Li, C., Singhal, V., & Rodrigues, R. (2010). A study of the internal and external effects of concurrency bugs. In 2010 IEEE/IFIP international conference on dependable systems and networks (DSN), pp. 221–230. IEEE.

Francesca, G., Santone, A., Vaglini, G., & Villani, M. L. (2011). Ant colony optimization for deadlock detection in concurrent systems. In Computer software and applications conference (COMPSAC), 2011 IEEE 35th Annual, pp. 108–117. IEEE.

Gao, Q., Zhang, W., Chen, Z., Zheng, M., & Qin, F. (2011). 2ndstrike: Toward manifesting hidden concurrency typestate bugs. *ACM SIGARCH Computer Architecture News*, *39*(1), 239–250.

Gesbert, L, Hu, Z, Loulergue, F, Matsuzaki, K, & Tesson, J (2010) Systematic development of correct bulk synchronous parallel programs. In 2010 international conference on parallel and distributed computing, applications and technologies (PDCAT), pp. 334–340. IEEE.

Godefroid, P. (1997). Model checking for programming languages using verisoft. In Proceedings of the 24th acm sigplan-sigact symposium on principles of programming languages (popl'97). New York, NY, USA: ACM.

Gottbrath, C. (2006). Eliminating parallel application memory bugs with totalview. In Proceedings of the 2006 ACM/IEEE conference on supercomputing, p. 210. ACM.

Gottschlich, J. E., Pokam, G. A., Pereira, C. L., & Wu, Y. (2013). Concurrent predicates: A debugging technique for every parallel programmer. In Proceedings of the 22nd international conference on parallel architectures and compilation techniques, pp. 331–340. IEEE Press.

Gupta, S., Sultan, F., Cadambi, S., Ivancic, F., & Rotteler, M. (2009). Using hardware transactional memory for data race detection. In IEEE international symposium on parallel & distributed processing, 2009. IPDPS 2009. pp. 1–11. IEEE.

Ha, O.-K., Kuh, I.-B., Tchamgoue, G. M., & Jun, Y.-K. (2012). On-the-fly detection of data races in OpenMP programs. In Proceedings of the 2012 workshop on parallel and distributed systems: Testing, analysis, and debugging, pp. 1–10. ACM.

Hilbrich, T., Protze, J., Schulz, M., de Supinski, B. R., & Mller, M. S. (2012). MPI runtime error detection with MUST: Advances in deadlock detection. In Proceedings of the international conference on high performance computing, networking, storage and analysis, 30. IEEE Computer Society Press.

Hong, S., & Kim, M. (2013). Effective pattern-driven concurrency bug detection for operating systems. *Journal of Systems and Software*, *86*(2), 377–388.

Hower, D. R., & Hill, M. D. (2008). Rerun: Exploiting episodes for lightweight memory race recording. In ACM SIGARCH computer architecture news, Vol. 36, pp. 265–276. IEEE computer society.

Huang, J., Meredith, P. O., & Rosu, G. (2014). Maximal sound predictive race detection with control flow abstraction. In Proceedings of the 35th ACM SIGPLAN conference on programming language design and implementation, p. 36. ACM.

Huang, J., & Bond, M. D. (2013). Efficient context sensitivity for dynamic analyses via calling context uptrees and customized memory management. In Proceedings of the 2013 ACM SIGPLAN international conference on object oriented programming systems languages & #38; applications. OOPSLA '13, pp. 53–72. New York, NY, USA: ACM. ISBN 978-1-4503-2374-1.

Huang, R., Halberg, E., & Suh, G. E. (2013). Non-race concurrency bug detection through order-sensitive critical sections. In ACM SIGARCH computer architecture news, Vol. 41, pp. 655–666. ACM.

Jalali, S., & Wohlin, C. (2012). Systematic literature studies: Database searches versus backward snowballing. In 2012 ACM-IEEE International symposium on empirical software engineering and measurement (ESEM), pp. 29–38, doi:10.1145/2372251.2372257.

Jannesari, A., & Tichy, W. F. (2008). On-the-fly race detection in multi-threaded programs. In Proceedings of the 6th workshop on parallel and distributed systems: Testing, analysis, and debugging, p. 6. ACM.

Jannesari, A., & Tichy, W. F. (2014). Library-independent data race detection. *Parallel and Distributed Systems, IEEE Transactions on*, *25*(10), 2606–2616.

Jin, G., Song, L., Zhang, W., Shan, L., & Liblit, B. (2011). Automated atomicity-violation fixing. *ACM SIGPLAN Notices*, *46*(6), 389–400.

Joshi, P., & Sen, K. (2008). Predictive typestate checking of multithreaded java programs. In Proceedings of the 2008 23rd IEEE/ACM international conference on automated software engineering, pp. 288–296. IEEE computer society.

Jyoti, A., & Arora, V. (2014). Debugging and visualization techniques for multithreaded programs: A survey. In Recent advances and innovations in engineering (ICRAIE), 2014, pp. 1–6. IEEE.

Kahlon, V. (2012). Automatic lock insertion in concurrent programs. In Formal methods in computer-aided design (FMCAD), 2012, pp. 16–23. IEEE.

Kahlon, V., Sankaranarayanan, S., & Gupta, A. (2013). Static analysis for concurrent programs with applications to data race detection. *International Journal on Software Tools for Technology Transfer*, *15*(4), 321–336.

Kahlon, V., Yang, Y., Sankaranarayanan, S., & Gupta, A. (2007). Fast and accurate static data-race detection for concurrent programs. In Computer aided verification, pp. 226–239. Springer.

Kang, M.-S., Ha, O.-K., & Jun, Y.-K. (2014). Visualization tool for debugging data races in structured fork-join parallel programs. *International Journal of Software Engineering and Its Applications*, *8*(4), 157–168.

Kasikci, B., Zamfir, C., & Candea, G. (2013). RaceMob: Crowdsourced data race detection. In Proceedings of the twenty-fourth ACM symposium on operating systems principles, pp. 406–422. ACM.

Keele, S. (2007). Guidelines for performing systematic literature reviews in software engineering, Technical report, Technical report, EBSE Technical Report EBSE-2007-01.

Kelly, T., Wang, Y., Lafortune, S., & Mahlke, S. (2009). Eliminating concurrency bugs with control engineering. *IEEE Computer*, *42*(11), 52–60.

Khoshavi, N., Zarandi, H.R., & Maghsoudloo, M. (2012). Two control-flow error recovery methods for multithreaded programs running on multi-core processors. In Microelectronics (MIEL), 2012 28th international conference on, pp. 371–374. IEEE.

Kiefer, K. E., & Moser, L. E. (2013). Replay debugging of non-deterministic executions in the Kernel-based virtual machine. *Software: Practice and Experience*, *43*(11), 1261–1281.

Kim, B.-C., & Jun, Y.-K. (2010). Program visualization for debugging deadlocks in multithreaded programs. In Advances in software engineering, pp. 228–236. Springer.

Kim, Y.-J., Lim, J.-S., & Jun, Y.-K. (2007a). Scalable thread visualization for debugging data races in OpenMP programs. In Advances in grid and pervasive computing, pp. 310–321. Springer.

Kim, Y.-J., Kang, M.-H., Ha, O.-K., & Jun, Y.-K. (2007b). Efficient race verification for debugging programs with openMP directives. In Parallel computing technologies, pp. 230–239. Springer.

Kistler, Michael, & Brokenshire, Daniel. (2011). Detecting race conditions in asynchronous DMA operations with full system simulation. In Performance analysis of systems and software (ISPASS), 2011 IEEE international symposium on, pp. 207–215. IEEE.

Li, H., Luo, J., & Li, W. (2014). A formal semantics for debugging synchronous message passing-based concurrent programs. *Science China Information Sciences*, *57*(12), 1–18.

Liu, P., & Zhang, C. (2012). Axis: Automatically fixing atomicity violations through solving control constraints. In Proceedings of the 34th international conference on software engineering, pp. 299–309. IEEE Press.

Lonnberg, J., Ben-Ari, Mordechai, & Malmi, Lauri. (2011). Visualising concurrent programs with dynamic dependence graphs. In Visualizing software for understanding and analysis (VISSOFT), 2011 6th IEEE international workshop on, pp. 1–4. IEEE.

Lu, K., Zhou, X., Wang, X., Zhang, W., & Li, G. (2013). RaceFree: An efficient multi-threading model for determinism. In ACM SIGPLAN notices, Vol. 48, pp. 297–298. ACM.

Lu, L., Ji, W., & Scott, M. L. (2014). Dynamic enforcement of determinism in a parallel scripting language. In Proceedings of the 35th ACM SIGPLAN conference on programming language design and implementation, p. 53. ACM.

Lu, S., Jiang, W., & Zhou, Y. (2007). A study of interleaving coverage criteria. In The 6th joint meeting on European software engineering conference and the ACM SIGSOFT symposium on the foundations of software engineering: companion papers, pp. 533–536. ACM.

Lu, S., Park, S., Seo, E., & Zhou, Y. (2008). Learning from mistakes: a comprehensive study on real world concurrency bug characteristics. In ACM Sigplan Notices, Vol. 43, pp. 329–339. ACM.

Lucia, B., & Ceze, L. (2009). Finding concurrency bugs with context-aware communication graphs. In Proceedings of the 42nd annual IEEE/ACM international symposium on microarchitecture, pp. 553–563. ACM.

Lucia, B., Ceze, L., & Strauss, K. (2010). ColorSafe: Architectural support for debugging and dynamically avoiding multi-variable atomicity violations. In ACM SIGARCH computer architecture news, Vol. 38, pp. 222–233. ACM.

Ma, H., Chen, Q., Wang, L., Liao, C., & Quinlan, D. (2012). An OpenMP analyzer for detecting concurrency errors. In Parallel processing workshops (ICPPW), 2012 41st international conference on, pp. 590–591. IEEE.

Machado, N., Romano, P., & Rodrigues, L. (2012). Lightweight cooperative logging for fault replication in concurrent programs. In 2012 42nd annual IEEE/IFIP international conference on dependable systems and networks (DSN), pp. 1–12. IEEE.

Maiya, P., Kanade, A., & Majumdar, R. (2014). Race detection for Android applications. In Proceedings of the 35th ACM SIGPLAN conference on programming language design and implementation, p. 34. ACM.

Makela, J.-M., Leppanen, V., & Forsell, M. (2013). Towards a parallel debugging framework for the massively multi-threaded, step-synchronous REPLICA architecture. In Proceedings of the 14th international conference on computer systems and technologies, pp. 153–160. ACM.

Martin, J.-P., Hicks, M., Costa, M., Akritidis, P., & Castro, M. (2010). Dynamically checking ownership policies in concurrent C/C++ programs. In ACM Sigplan Notices, Vol. 45, pp. 457–470. ACM.

Moiseev, M., Glukhikh, M., Zakharov, A., & Richter, H. (2013). A static analysis approach to data race detection in systemic designs. In 2013 IEEE 16th international symposium on design and diagnostics of electronic circuits & systems (DDECS), pp. 54–59. IEEE.

Montesinos, P., Ceze, L., & Torrellas, J. (2008). Delorean: Recording and deterministically replaying shared-memory multiprocessor execution ef? ciently. In 35th International symposium on computer architecture, 2008. ISCA'08, pp. 289–300. IEEE.

Mozaffari-Kermani, M., Azarderakhsh, R., Lee, C.-Y., & Bayat-Sarmadi, S. (2014). Reliable concurrent error detection architectures for extended euclidean-based division over gf(m2). Very large scale integration (VLSI) systems. *IEEE Transactions on*, *22*(5), 995–1003.

Nanz, S., Torshizi, F., Pedroni, M., & Meyer, B. (2013). Design of an empirical study for comparing the usability of concurrent programming languages. *Information and Software Technology*, *55*(7), 1304–1315.

Negishi, Y., Murata, H., Cong, G., Wen, H.-F., Chung, I. et al. (2012). A static analysis tool using a three-step approach for data races in HPC programs. In Proceedings of the 2012 workshop on parallel and distributed systems: Testing, analysis, and debugging, pp. 11–17. ACM.

Oßner, C., & Böhm, K. (2013). Graphs for mining-based defect localization in multithreaded programs. *International Journal of Parallel Programming*, *41*(4), 570–593.

Park, C.-S., & Sen, K. (2008). Randomized active atomicity violation detection in concurrent programs. In Proceedings of the 16th ACM SIGSOFT international symposium on foundations of software engineering, pp. 135–145. ACM.

Park, C.-S., & Sen, K. (2012). Concurrent breakpoints. In ACM SIGPLAN notices, Vol. 47, pp. 331–332. ACM.

Park, H.-D., & Jun, Y.-K. (2012). Detecting first races in shared-memory parallel programs with random synchronization. In Computer applications for graphics, grid computing, and industrial environment, pp. 165–169. Springer.

Park, M.-Y., & Chung, S.-H. (2008). Detection of first races for debugging message-passing programs. In Computer and information technology, 2008. CIT 2008. 8th IEEE international conference on, pp. 261–266. IEEE.

Park, M.-Y., Kim, S. Y., & Park, H.-R. (2007). Visualization of affect-relations of message races for debugging MPI programs. In IEEE international conference on granular computing, 2007. GRC 2007. pp. 745–745. IEEE.

Park, S. (2013). Debugging non-deadlock concurrency bugs. In Proceedings of the 2013 international symposium on software testing and analysis, pp. 358–361. ACM.

Park, S., Vuduc, R., & Harrold, M. J. (2012). A unified approach for localizing non-deadlock concurrency bugs. In 2012 IEEE Fifth International Conference on software testing, verification and validation (ICST), pp. 51–60. IEEE.

Petersen, K., Feldt, R., Mujtaba, S., & Mattsson, M. (2008). Systematic mapping studies in software engineering. In 12th International conference on evaluation and assessment in software engineering, Vol. 17.

Prvulovic, M. (2006). CORD: Cost-effective (and nearly overhead-free) order-recording and data race detection. In The twelfth international symposium on high-performance computer architecture, 2006, pp. 232–243. IEEE.

Pun, K. I., Steffen, M., & Stolz, V. (2014). Deadlock checking by data race detection. *Journal of Logical and Algebraic Methods in Programming*, *83*(5), 400–426.

Qi, S., Otsuki, N., Nogueira, L. O., Muzahid, A., & Torrellas, J. (2012). Pacman: Tolerating asymmetric data races with unintrusive hardware. In 2012 IEEE 18th International symposium on high performance computer architecture, pp. 1–12. IEEE.

Qi, Y., Das, R., Luo, Z. D., & Trotter, M. (2009). Multicoresdk: A practical and efficient data race detector for real-world applications. In Proceedings of the 7th workshop on parallel and distributed systems: Testing, analysis, and debugging, p. 5. ACM.

Raychev, V., Vechev, M., & Sridharan, M. (2013). Effective race detection for event-driven programs. In ACM SIGPLAN notices, Vol. 48, pp. 151–166. ACM.

Rister, B. D., Campbell, J., Pillai, P., & Mowry, T. C. (2007). Integrated debugging of large modular robot ensembles. In 2007 IEEE international conference on robotics and automation, pp. 2227–2234. IEEE.

Rossi, D., Omaa, M., Garrammone, G., Metra, C., Jas, A., & Galivanche, R. (2013). Low cost concurrent error detection strategy for the control logic of high performance microprocessors and its application to the instruction decoder. *Journal of Electronic Testing*, *29*(3), 401–413.

Sack, P., Bliss, B. E., Ma, Z., Petersen, P., & Torrellas, J. (2006). Accurate and efficient filtering for the intel thread checker race detector. In Proceedings of the 1st workshop on Architectural and system support for improving software dependability, pp. 34–41. ACM.

Sadowski, C., & Yi, J. (2009). Tiddle: A trace description language for generating concurrent benchmarks to test dynamic analyses. In Proceedings of the seventh international workshop on dynamic analysis, pp. 15–21. ACM.

Said, M., Wang, C., Yang, Z., & Sakallah, K. (2011). Generating data race witnesses by an SMT-based analysis. In NASA formal methods, pp. 313–327. Springer.

Schaeli, B., & Hersch, R. D. (2008). Dynamic testing of flow graph based parallel applications. In Proceedings of the 6th workshop on parallel and distributed systems: Testing, analysis, and debugging, p. 2. ACM.

Schneider, J. (2014). Tracking down root causes of defects in simulink models. In Proceedings of the 29th ACM/IEEE international conference on Automated software engineering. ACM.

Schuppan, V., Baur, M., & Biere, A. (2005). JVM independent replay in Java. *Electronic Notes in Theoretical Computer Science*, *113*, 85–104. doi:10.1016/j.entcs.2004.01.032.

Sen, K. (2008). Race directed random testing of concurrent programs. In ACM SIGPLAN notices, Vol. 43, pp. 11–21. ACM.

Serebryany, K., & Iskhodzhanov, T. (2009). ThreadSanitizer: Data race detection in practice. In Proceedings of the workshop on binary instrumentation and applications, pp. 62–71. ACM.

Shimomura, T., & Ikeda, K. (2013). Waiting blocked-tree type deadlock detection. In Science and information conference (SAI), 2013, pp. 45–50. IEEE.

Shousha, M., Briand, L., & Labiche, Y. (2012). A uml/marte model analysis method for uncovering scenarios leading to starvation and deadlocks in concurrent systems. *Software Engineering, IEEE Transactions on*, *38*(2), 354–374.

Shousha, M., B., Lionel C., & Labiche, Y. (2009). A uml/marte model analysis method for detection of data races in concurrent systems. In Model driven engineering languages and systems, pp. 47–61. Springer.

Song, Y. W., & Lee, Y.-H. (2014). Efficient data race detection for C/C++ programs using dynamic granularity. In Parallel and distributed processing symposium, 2014 IEEE 28th international, pp. 679–688. IEEE.

Tallam, S., Tian, C., & Gupta, R. (2008). Dynamic slicing of multithreaded programs for race detection. In IEEE International conference on software maintenance, 2008. ICSM 2008, pp. 97–106. IEEE.

Tan, L., Feng, M., & Gupta, R. (2013). Lightweight fault detection in parallelized programs. In 2013 IEEE/ACM International symposium on code generation and optimization (CGO), pp. 1–11. IEEE.

Tchamgoue, G.M., Gan, L., Ha, O.-K., Yang, S.-W., & Jun, Y.-K. (2012). Visualizing concurrency faults in ARINC-653 real-time applications. In Digital avionics systems conference (DASC), 2012 IEEE/AIAA 31st, pp. 9–61. IEEE.

Tchamgoue, G. M., Kuh, I.-B., Ha, O.-K., Kim, K.-H., & Jun, Y.-K. (2010). A race healing framework in simulated ARINC-653. In Communication and networking, pp. 238–246. Springer.

Teixeira, B., Loureno, J., Farchi, E., Dias, R., & Sousa, D. (2010). Detection of transactional memory anomalies using static analysis. In Proceedings of the 8th workshop on parallel and distributed systems: Testing, analysis, and debugging, pp. 26–36. ACM.

Tian, C., Nagarajan, V., Gupta, R., & Tallam, S. (2009). Automated dynamic detection of busywait synchronizations. *Software: Practice and Experience*, *39*(11), 947–972.

Torrellas, J., Ceze, L., Tuck, J., Cascaval, C., Montesinos, P., Ahn, W., et al. (2009). The Bulk Multicore architecture for improved programmability. *Communications of the ACM*, *52*(12), 58–65.

Trainin, E., Nir-Buchbinder, Y., Tzoref-Brill, R., Zlotnick, A., Ur, S., & Farchi, E. (2009). Forcing small models of conditions on program interleaving for detection of concurrent bugs. In Proceedings of the 7th workshop on parallel and distributed systems: Testing, analysis, and debugging, p. 7. ACM.

Tzoref, R., Ur, S., & Yom-Tov, E., (2007). Instrumenting where it hurts: An automatic concurrent debugging technique. In Proceedings of the 2007 international symposium on software testing and analysis, pp. 27–38. ACM.

Uhrig, S. (2011). Tracing static fields of embedded parallel Java applications. In Computer software and applications conference workshops (COMPSACW), 2011 IEEE 35th annual, pp. 516–519. IEEE.

Vasudevan, N., Edwards, S. A., & Singh, S., (2008). A deterministic multi-way rendezvous library for Haskell. In IPDPS 2008. IEEE international symposium on parallel and distributed processing, 2008, pp. 1–12. IEEE.

Veeraraghavan, K., Chen, P. M., Flinn, J., & Narayanasamy, S. (2011). Detecting and surviving data races using complementary schedules. In Proceedings of the twenty-third ACM symposium on operating systems principles, pp. 369–384. ACM.

Viennot, N., Nair, S., & Nieh, J. (2013). Transparent mutable replay for multicore debugging and patch validation. In ACM SIGARCH computer architecture news, Vol. 41, pp. 127–138. ACM.

Vo, A., & Gopalakrishnan, G. (2010). Scalable verification of MPI programs. In2010 IEEE international symposium on parallel & distributed processing, workshops and Phd forum (IPDPSW), pp. 1–4. IEEE.

Wang, J.-Y., Shue, Y.-S., & Bagchi, S. (2007). Pesticide: Using SMT to improve performance of pointer-bug detection. In International conference on computer design, 2006. ICCD 2006, pp. 514–521. IEEE.

Wang, L., & Stoller, S. D. (2006a). Accurate and efficient runtime detection of atomicity errors in concurrent programs. In Proceedings of the eleventh ACM SIGPLAN symposium on Principles and practice of parallel programming, pp. 137–146. ACM.

Wang, L., & Stoller, S. D. (2006b). Runtime analysis of atomicity for multithreaded programs. *Software Engineering, IEEE Transactions on*, *32*(2), 93–110.

Wang, N., Han, J., & Fang, J. (2012a). A transparent control-flow based approach to record-replay non-deterministic bugs. In 2012 IEEE 7th international conference on networking, architecture and storage (NAS), pp. 189–198. IEEE.

Wang, P., Zhang, X., Hao, P., & Zhang, Y. (2012b). Towards the multithreaded deterministic replay in program debugging. In 2012 8th international conference on information science and digital content technology (ICIDT), Vol. 1, pp. 139–144. IEEE.

Wang, T., Shen, L., & Ma, C. (2014). A process algebra-based detection model for multithreaded programs in communication system. *KSII Transactions on Internet and Information Systems (TIIS)*, *8*(3), 965–983.

Wang, W., & Fang, B. (2005). Replaying message-passing programs with an efficient logical clock. *WSEAS Transactions on Computers*, *4*(7), 750–757.

Wang, W., Wang, Z., Wu, C., Yew, P.-C., Shen, X., Yuan, X., Li, J., Feng, X., & Guan, Y. (2014). Localization of concurrency bugs using shared memory access pairs. In Proceedings of the 29th ACM/IEEE international conference on Automated software engineering, pp. 611–622. ACM.

Wang, Y., Liu, P., Kelly, T., Lafortune, S., Reveliotis, S. A., & Zhang, C. (2012). On atomicity enforcement in concurrent software via discrete event systems theory. In CDC, pp. 7230–7237.

Watson, G. R., Rasmussen, C. E., Tibbitts, B. R. (2009). An integrated approach to improving the parallel application development process. In IEEE International Symposium on parallel & distributed processing, 2009. IPDPS 2009, pp. 1–8. IEEE.

Weeratunge, D., Zhang, X., & Jagannathan, S. (2010a). Analyzing multicore dumps to facilitate concurrency bug reproduction. *ACM Sigplan Notices*, *45*(3), 155–166.

Weeratunge, D., Zhang, X., Sumner, W. N., & Jagannathan, S. (2010b). Analyzing concurrency bugs using dual slicing. In Proceedings of the 19th international symposium on Software testing and analysis, pp. 253–264. ACM.

Wen, C.-N., Chou, S.-H., & Chen, T.-F. (2009). dIP: A non-intrusive debugging IP for dynamic data race detection in many-core. In 2009 10th International symposium on pervasive systems, algorithms, and networks (ISPAN), pp. 86–91. IEEE.

Wen, C.-N., Chou, S.-H., Chen, C.-C., & Chen, T.-F. (2012). NUDA: A non-uniform debugging architecture and nonintrusive race detection for many-core systems. *Computers, IEEE Transactions on*, *61*(2), 199–212.

Wen, Y., Zhao, J., Huang, M., & Chen, H. (2011). Towards detecting thread deadlock in java programs with jvm introspection. In Trust, security and privacy in computing and communications (TrustCom), 2011 IEEE 10th International conference on, pp. 1600–1607. IEEE.

Wester, B., Devecsery, D., Chen, P. M., Flinn, J., & Narayanasamy, S. (2013). Parallelizing data race detection. In ACM SIGARCH computer architecture news, Vol. 41, pp. 27–38. ACM.

Wieringa, R., Maiden, N., Mead, N., & Rolland, C. (2005). Requirements engineering paper classification and evaluation criteria: A proposal and a discussion. *Requirements Engineering*, *11*(1), 102–107.

Wu, X., Wei, J., & Wang, X. (2012). Debug concurrent programs with visualization and inference of event structure. In Software engineering conference (APSEC), 2012 19th Asia-Pacific, Vol. 1, pp. 683–692. IEEE.

Wu, X., Wen, Y., Chen, L., Dong, W., & Wang, J., (2013). Data race detection for interrupt-driven Programs via bounded model checking. In Software security and reliability-companion (SERE-C), 2013 IEEE 7th international conference on, pp. 204–210. IEEE.

Yoshiura, N., & Wei, W. (2014). Static data race detection for Java programs with dynamic class loading. In Internet and distributed computing systems, pp. 161–173. Springer.

Yu, J., Ci, Y., Zhou, P., Wu, Y., & Zhao, C. (2012). Deterministic replay of multithread applications using virtual machine. In Advanced information networking and applications workshops (WAINA), 2012 26th International conference on, pp. 429–434. IEEE.

Yuan, D., Mai, H., Xiong, W., Tan, L., Zhou, Y., & Pasupathy, S., (2010). SherLog: Error diagnosis by connecting clues from run-time logs. In ACM SIGARCH computer architecture news, Vol. 38, pp. 143–154. ACM.

Zaineb, G., & Manarvi, I. A. (2011). Identification and analysis of causes for software bug rejection with their impact over testing efficiency. International journal of software engineering & applications (IJSEA) 2 (4). http://airccse.org/journal/ijsea/papers/1011ijsea07.pdf.

Zeller, A. (2009). *Why programs fail: A guide to systematic debugging*. Amsterdam: Elsevier.

Zhang, W., De Kruijf, M., Li, A., Shan, L., & Sankaralingam, Karthikeyan. (2013). ConAir: Featherweight concurrency bug recovery via single-threaded idempotent execution. *ACM SIGARCH Computer Architecture News*, *41*(1), 113–126.

Zhou, P., Teodorescu, R., & Zhou, Y. (2007). HARD: Hardware-assisted lockset-based race detection. In IEEE 13th International symposium on high performance computer architecture, 2007. HPCA 2007, pp. 121–132. IEEE.

Zyulkyarov, F., Harris, T., Unsal, O. S., Cristal, A., & Valero, M. (2010). Debugging programs that use atomic blocks and transactional memory. In ACM sigplan notices, Vol. 45, pp. 57–66. ACM.

Sara Abbaspour Asadollah is a Ph.D. student at Mälardalen University, Sweden. She has completed her Master's Degree in Software Engineering at Faculty of Computer Science and Information, University of Malaya, Malaysia. The field of her research during the Master's Degree was Web Engineering. She obtained her Bachelor's Degree in Computer Engineering from Iran. Her research area was focused on Image Processing. She also has work experience in various aspects of industrial environment such as Mobile Development Systems, Multimedia Technologies and ELearning application, RFID and Smart Cards Technologies, and Software System Testing.

Daniel Sundmark is a part-time senior lecturer and assistant professor at IDT and a researcher in the Software Testing Laboratory. He received a M.Sc. degree (civilingenjör) in Information Technology from Uppsala University in 2003, a licentiate degree from Mälardalen University in 2004, and a Ph.D. degree from Mälardalen University in 2008. Daniel shares his time between MDH and the Swedish Institute of Computer Science (SICS).

Sigrid Eldh started at Uppsala University 1981 where she took her Masters in Computer Science (DVL) as well as some other courses (pedagogical, judicial, economy, pedagogy, English). Also worked with creating courses in Operating systems, programming languages (Pascal), New Systems and Architectures and Databases. Started her company 84. After several jobs she has ended up at Ericsson (started-94). Worked at HP in Melbourne (OZ) 96–97 and back at Ericsson in different positions. She has a genuine interest for software testing and verification, processes, methods, techniques, quality, measurement, improvements, leadership, personal development and management.

Hans Hansson is professor in Real-Time Systems at Mälardalen University since 1997. He is director of Mälardalen Real-Time Research Centre and the PROGRESS national strategic research centre, Scientific Leader of SICS Swedish ICT Västerås AB and the EU/ARTEMIS project SafeCer. He received an M.Sc. (Engineering Physics), a Licentiate degree (Computer Systems), a BA (Business Administration), and a Doctor of Technology degree (Computer Systems) from Uppsala University (UU), Sweden, in 1981, 1984, 1984 and 1992. Prof. Hansson's previous appointments include being director of the nat'l research programme ARTES, visiting prof. and dept. chair at the Dept. of Computer Systems, Uppsala University, and researcher and scientific advisor at the Swedish Institute of Computer Science (SICS) in Stockholm.

Wasif Afzal completed his M.Sc., Licentiate and Ph.D. in Software Engineering from Blekinge Institute of Technology (BTH), Sweden, in 2007, 2009 and 2011, respectively. He was a postdoctoral researcher at BTH between September 2011 and March 2012. From September 2011 to August 2013, he was also an assistant professor in software engineering at Bahria University, Islamabad, Pakistan.