Information and Software Technology 124 (2020) 106312

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Multiple fault localization of software programs: A systematic literature review

Abubakar Zakari[a,*], Sai Peck Lee[b], Rui Abreu[d], Babiker Hussien Ahmed[b], Rasheed Abubakar Rasheed[c]

[a] Department of Computer Science, Kano University of Science and Technology, Wudil, P.M.B 3244, Kano, Nigeria
[b] Department of Software Engineering, Faculty of Computer Science and Information Technology, University of Malaya, 50603, Kuala Lumpur, Malaysia
[c] Department of Computer System & Technology, Faculty of Computer Science and Information Technology, University of Malaya, 50603, Kuala Lumpur, Malaysia
[d] INESC-ID and IST, University of Lisbon, Portugal

A R T I C L E I N F O

Keywords:
Program debugging
Parallel debugging
Fault localization
Multiple faults
One-bug-at-a-time (OBA)

A B S T R A C T

Context: Multiple fault localization (MFL) is the act of identifying the locations of multiple faults (more than one fault) in a faulty software program. This is known to be more complicated, tedious, and costly in comparison to the traditional practice of presuming that a software contains a single fault. Due to the increasing interest in MFL by the research community, a broad spectrum of MFL debugging approaches and solutions have been proposed and developed.
Objective: The aim of this study is to systematically review existing research on MFL in the software fault localization (SFL) domain. This study also aims to identify, categorize, and synthesize relevant studies in the research domain.
Method: Consequently, using an evidence-based systematic methodology, we identified 55 studies relevant to four research questions. The methodology provides a systematic selection and evaluation process with rigorous and repeatable evidence-based studies selection process.
Result: The result of the systematic review shows that research on MFL is gaining momentum with stable growth in the last 5 years. Three prominent MFL debugging approaches were identified, i.e. One-bug-at-a-time debugging approach (OBA), parallel debugging approach, and multiple-bug-at-a-time debugging approach (MBA), with OBA debugging approach being utilized the most.
Conclusion: The study concludes with some identified research challenges and suggestions for future research. Although MFL is becoming of grave concern, existing solutions in the field are less mature. Studies utilizing real faults in their experiments are scarce. Concrete solutions to reduce MFL debugging time and cost by adopting an approach such as MBA debugging approach are also less, which require more attention from the research community.

1. Introduction

Software plays an important role in our daily life. However, the increasing scale and complexity of software in the last decades has a major challenge to its development and diagnosis [1]. Therefore, identifying the exact locations of software faults with accuracy and speed is a continues challenge to software developers and researchers. Faults manifestation can be discovered in software due to erroneous behavior, however, finding and fixing these faults is an entirely different issue. Fault localization focuses on the latter (finding the location of faults). When a software program fails, this failure can be caused by many faults that may reside in the program [2-4]. However, in the literature, various studies made a single fault assumption (a program failure is caused by a single fault) [5-8]. Localization of multiple faults has proven to be more difficult, time-consuming, and costly due to the issue of fault interference that leads to the reduction in effectiveness of the existing fault localization techniques [9].

In the last 10 years, a significant number of studies have been published to address the issue of multiple fault localization (MFL) by contributing new and different solutions on ways to understand faults-to-failures relationships and localize the faults effectively. Various review and survey papers such as [10-16] have been published in software fault localization (SFL) research domain. Wong et al. conducted a detail general survey in the SFL research domain [10]. Recently, Zakari et al. also

* Corresponding author.
E-mail address: abubakar.zakari@yahoo.com (A. Zakari).

https://doi.org/10.1016/j.infsof.2020.106312
Received 7 May 2019; Received in revised form 18 March 2020; Accepted 20 March 2020
Available online 25 March 2020

Table 1
Research questions.

RQ #	Research question	Motivation
RQ1	What are the selected studies demographics and characteristics?	To highlight the relevant publication channels, relevant publication sources, the frequency of publication per year, and presents the selected studies based on their Quality assessment scores.
RQ2	What are the debugging approaches widely used to localize multiple faults?	To identify the widely used debugging approaches in localizing multiple faults.
RQ3	What are the datasets, datasets programming languages, and fault types used in localizing multiple faults?	To highlights the current datasets that are widely used in the localization of multiple faults. To provide the commonly used fault types and the programming languages (with respect to datasets) used by each study in the localization of multiple faults.
RQ4	What are the evaluation metrics used in multiple fault localization (MFL)?	To highlights the most utilized evaluation metrics used in the context of multiple faults.

published a systematic mapping study on SFL to primarily identify the general trends and productivity in the SFL research domain [17]. However, looking at all the papers, there is no systematic literature review (SLR) that specifically targets literature on addressing multiple fault localization. This SLR tempts to fill this research gap by extensively identifying, categorizing, and synthesizing relevant studies in MFL research domain.

In this paper, an evidence-based systematic methodology is used to make sure all relevant studies relating to MFL in the last decade (2009 - 2018) are identified and retrieved. This methodology provides a systematic selection and evaluation process with rigorous and repeatable evidence-based studies selection process. This SLR also presents results based on the selected studies general demographics and characteristics, the MFL debugging approaches currently used in the literature, the datasets used by selected studies, datasets programming languages, the type of faults used (real or artificial faults), and the evaluation metrics used in evaluating proposals on MFL research. The contributions of the study are as follows:

- A comprehensive systematic review on multiple fault localization (MFL).
- Analyzing and synthesizing existing works in this research domain.
- Identifying current research challenges and areas that lacks and needs attention from the software research community in this domain.

The paper is structured as follows. Section 2 outlined the systematic research methodology. Section 3 gives the detailed results of our findings in relation to the respective research questions. Section 4 provides a discussion with regards to the results of the study. Lastly, the paper is concluded in Section 5.

2. Research method

A Systematic literature review (SLR) is conducted solely to identify, evaluate, interpret, and also report research related to a specific field of interest [18]. This SLR adopts an evidence-based searching and study selection procedures to improve transparency. Therefore, a SLR must be conducted with a rigorous search plan that is transparent, fair, and unbiased. The search strategy or plan must ensure the completeness of the search for assessment [19]. At the time of this report, there was no systematic literature study that provided a rigorous review and analysis of existing research on MFL [10-16]. Therefore, our main aim is to fill this research gap by conducting an SLR using Kitchenham's methodology [20]. The existing systematic review process composed of various steps that must be performed or conducted in a systematic and disciplined way, which includes developing a review protocol, conducting a systematic review, analyzing the obtained results, reporting the results, visualizing the results, and discussing the findings.

Table 2
Electronic databases.

Database ID	Database name	Link
ED1	ACM	http://dl.acm.org/
ED2	IEEE Xplore	http://ieeexplore.ieee.org/
ED3	Science Direct	http://sciencedirect.com/
ED4	Springer Link	http://link.springer.com/
ED5	Wiley	http://onlinelibrary.wiley.com/

2.1. Research questions

The overall objective of this SLR is to gain insight into studies based on MFL under the domain of SFL. In order to have a detailed view of this topic, the SLR addresses four key research questions (RQs). These key RQs will allow us to categorize and fully understand the current research in MFL and to identify limitations and future research directions in the field. The key RQs are listed in Table 1.

2.2. Data sources

Table 2 highlights the five electronic databases adopted for this study. These databases were considered as primary data sources for retrieving potentially relevant studies. However, we excluded Google Scholar due to low precision results and overlapping of results from other data sources adopted in Table 2. Therefore, all relevant articles in Google Scholar are already captured by the other sources.

2.3. Search terms

To adequately and effectively search for relevant studies, relevant search terms are crucial. Keele [18] has suggested population, intervention, comparison, and outcome (PICO) viewpoints in this regard. These viewpoints have been broadly used by several SLRs and Systematic mapping studies [17, 21, 22]. The relevant terms for population, intervention, and outcome are as follows.

Population: Software, Program
Intervention: Fault, Bug, defect
Outcome: localization, diagnosis

On the basis of PICO structure, a generic Search string was constructed to maintain the consistency of search along multiple databases. Therefore, to perform the automatic search in the electronic databases (Table 2), the following Search string was utilized.

Generic: ((Software OR program) AND (fault OR bug OR defect) AND (localize OR localization OR diagnosis Or diagnose))

The following search terms are based on key terms identified from the SFL research domain which were also used in our former study [17].

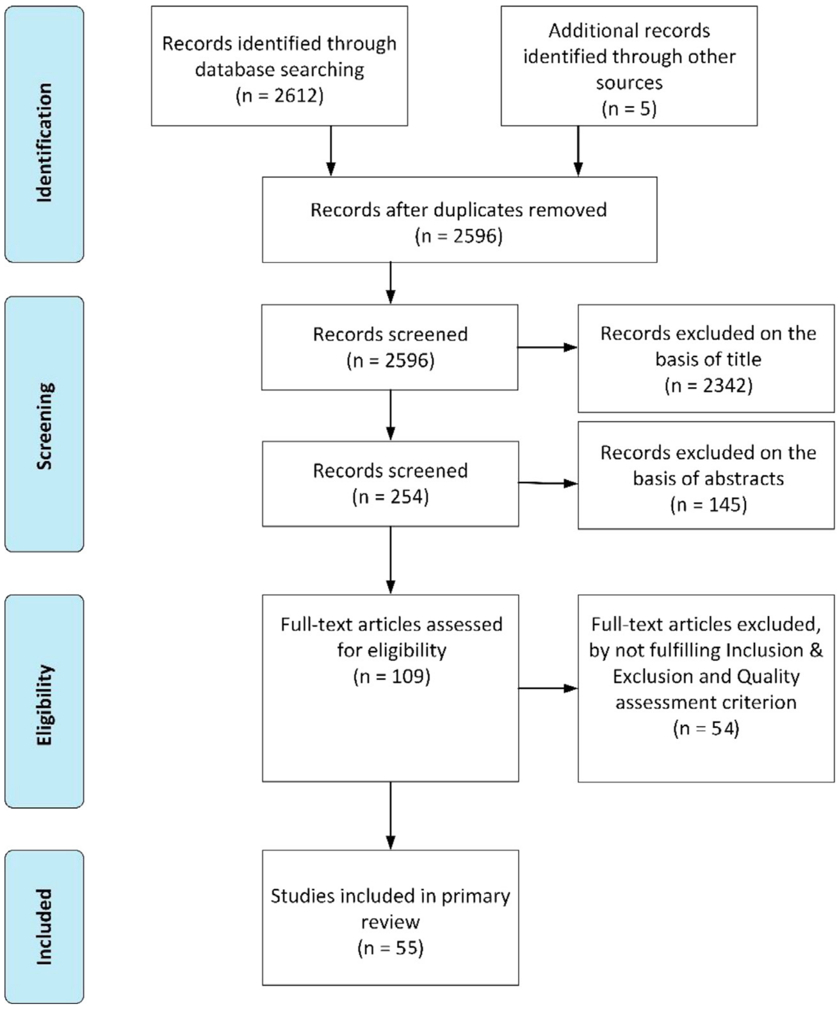

Identification
Screening
Eligibility
Included
Records identified through database searching (n = 2612)
Additional records identified through other sources (n = 5)
Records after duplicates removed (n = 2596)
Records screened (n = 2596)
Records excluded on the basis of title (n = 2342)
Records screened (n = 254)
Records excluded on the basis of abstracts (n = 145)
Full-text articles assessed for eligibility (n = 109)
Full-text articles excluded, by not fulfilling Inclusion & Exclusion and Quality assessment criterion (n = 54)
Studies included in primary review (n = 55)

Fig. 1. Study selection procedure.

These search terms were defined to search for relevant articles from the selected electronic databases listed in Table 2. However, as different databases have different interfaces for advanced search and command search, this particular search string was used on the five databases.

2.4. *Study selection procedure*

The aim of the study selection process was to identify relevant articles that are most relevant to the objectives of this SLR study. Fig. 1 visualized the study selection procedure of our systematic review. The process composed of three phases, each of which was achieved by a thorough consensus meeting between the respective researchers to ensure more confidence and minimal bias in the selection process. Therefore, if the same article appeared in multiple sources, only one will be considered based on our search order. The first author, the second author, and the third author integrated the search results for different searchers and also undertook an initial screening of the 2617 articles found, based on their title, abstract, and conclusion. For each article initially screened, it was evaluated by two researchers to further decide whether if the article should be included. Therefore, articles that were judged differently were discussed by the two researchers who carried out the evaluation of articles until an agreement was found. The remaining researchers, the fourth author and the fifth author also partake in reviewing the final selected articles. This screening was to exclude studies that were obviously irrelevant, duplicate, or studies that did not address the issue of multiple faults localization.

2.5. *Inclusion and exclusion criteria*

To answer the research questions in our SLR, inclusion (IC) and exclusion (EC) criteria were formulated and utilized to aid in selecting potential studies from the data sources. These criteria were applied to all the studies retrieved in the distinct phases of the study selection procedure (as shown in Fig. 1). For article search, the time period was set from January 2009 to December 2018 (10 years period), to make sure that only recent, up-to-date articles were included and also to retrieved studies that were published in the last 10 years. Also, early cited articles were included, provided the full text was available.

The IC and EC criteria employed in this SLR are listed in Table 3. These criteria were applied to the 2nd and 3rd phases of the study selection procedure (Fig. 1). For the 2nd phase, the IC and EC criteria were considered based on the articles' title, abstracts, and conclusions. Therefore, 109 out of 254 articles were filtered through the 2nd phase. For the 3rd phase, In order to increase confidence on articles coverage, a snowballing procedure was applied on 109 full-text articles assessed. We conducted backward and forward snowballing. Firstly, in backward snowballing, we go through article's reference list and exclude articles that do not fulfill our basic criteria. In forward snowballing, we analyzed our articles based on the articles' citing the article being examined. Therefore, each article citing a given article is examined. Hence, the inclusion and exclusion of articles were considered on the basis of the IC and EC criteria in Table 3 and quality attributes in Section 2.6. Both criteria were applied simultaneously to the full texts of all the 109 articles. Finally, 55 articles were selected for the final list. Fig. 2 presents the pro-

Table 3
Inclusion and exclusion criteria.

	Inclusion criteria
IC1	Peer-reviewed articles
IC2	Articles providing multiple fault localization (MFL) techniques/approaches in the context of software systems only
IC3	Articles providing techniques/approaches to enhance or support multiple fault localization (MFL).
IC4	Inclusion of articles that consider localizing faults on multiple-fault programs
IC5	Inclusion of articles published in the last ten years only (2009–2018)
	Exclusion Criteria
EC1	Studies other than the English language
EC2	Fault localization techniques in the context other than software systems
EC3	Articles that exclusively consider localization on single-fault programs
EC4	Studies on software fault prediction
EC5	Articles providing unclear results and findings
EC6	Articles with no validation of proposed techniques/approaches
EC7	Validation through expert opinion

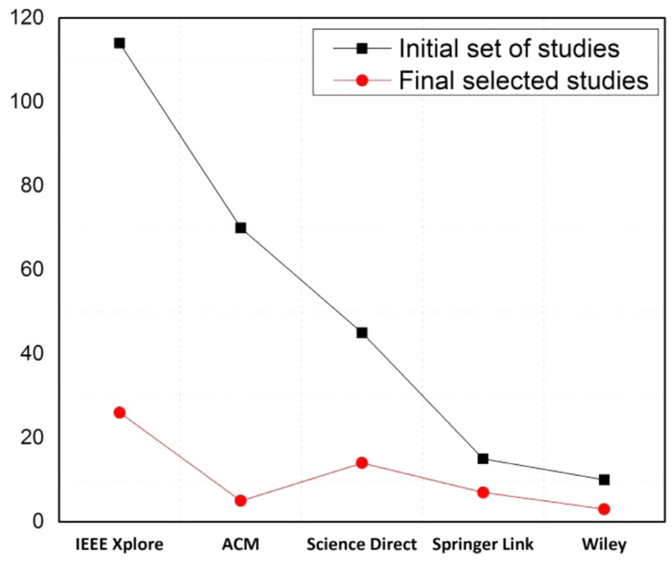


Fig. 2. The proportion of selected studies.

portion with respect to the initial number of articles in correspondence to the final selected articles from each electronic data source listed in Table 2.

2.6. Quality assessment criteria

Quality assessment (QA) is vital in every SLR. Hence, the QA process was performed during the 3rd phase of the study selection process. The inclusion/exclusion with QA criteria was applied to the articles filtered in the 2nd phase of the study selection. 109 articles were received by the researchers in the 3rd phase where each article was scrutinized by the researchers to remove bias.

Therefore, a questionnaire was designed to assess the quality of the selected articles. The process in which this questionnaire was formulated was inspired by a previous systematic study [21].

1) **QC1:** The study provides a contribution towards how multiple faults can be localized effectively. The potential answers were “Yes (+1)”, “Partially (+0.5)”, and “No (+0)”.
2) **QC2:** The goals and objectives of the study are clearly defined. The potential answer was “Yes (+1)”, “Partially (+0.5)”, and “No (+0)”.
3) **QC3:** The study presents empirical results with rigorous evaluation. The potential answers were “Yes (+1)”, “Partially (+0.5)”, and “No (+0)”.
4) **QC4:** Contribution and limitation are clearly stated. The potential answers were “Yes (+1)” and “No (+0)”.
5) **QC5:** There are minimal biases in the study. The potential answers were “Yes (+1)” and “No (+0)”.
6) **QC6:** The study has been published in a recognized and stable publication source. To answer this question, we considered the computer science conference ranking (CORE) [23] (A, B, and C), and Journal Citation Reports (JCR) lists.

Therefore, the potential answers were as follow:

- Conferences, workshops, and symposium:
  - (+1.5) if ranked CORE A,
  - (+1) if ranked CORE B,
  - (+0.5) if ranked CORE C,
  - (+0) if not in CORE ranking.
- Journals:
  - (+2) if ranked Q1,
  - (+1.5) if ranked Q2,
  - (+1) if ranked Q3 or Q4,
  - (+0) if it has no JCR ranking.
- Others; (+0).

The score for the quality criterion in (QC6) shows the obvious fact that journals carry more weight than Conferences, Workshops, and Symposiums due to the fact that the likelihood of publishing a work in Q1 or Q2 journal may be difficult than other publication channels. Therefore, a scale of 1–7 served as the final quality score for a given article.

2.7. Data extraction

The selected articles after the 2nd phase of the study selection procedure were analyzed by the review team. The full text of each article was analyzed by at least two researchers. Therefore, important information was extracted to a predefined data extraction form. The form composed of the following list of items.

- Title
- Publication venue
- Publication year
- Type of solution proposed
- Debugging approach utilized
- Research type method (solution paper, evaluation paper, experience paper, validation paper, others)
- Dataset used
- Programming language of the dataset used
- Fault type (artificial faults or real faults)
- Evaluation metric utilized

3. Results

In this section, the results of the study with respect to the individual research questions (RQs) are presented.

Table 4
Overview of selected studies.

Identifier	Study reference	Publication year	Publication channel	Citation count	Research type	SFL technique
S1	[24]	2013	Conference	4	Solution paper	Statistical-based
S2	[25]	2014	Conference	11	Solution paper	Spectrum-based
S3	[26]	2016	Conference	20	Solution paper	Spectrum-based
S4	[27]	2011	Symposium	82	Evaluation paper	Spectrum-based
S5	[28]	2010	Conference	12	Solution paper	Hybrid
S6	[29]	2016	Conference	1	Solution paper	Spectrum-based
S7	[2]	2015	Journal	24	Evaluation paper	Spectrum-based
S8	[30]	2016	Journal	16	Solution paper	Misc.
S9	[31]	2018	Journal	3	Solution paper	Spectrum-based
S10	[32]	2018	Journal	2	Solution paper	Spectrum-based
S11	[33]	2017	Journal	11	Evaluation paper	Spectrum-based
S12	[34]	2016	Journal	6	Solution paper	Spectrum-based
S13	[35]	2016	Journal	1	Solution paper	Hybrid
S14	[36]	2016	Journal	2	Solution paper	Misc.
S15	[37]	2016	Journal	6	Solution paper	Misc.
S16	[38]	2015	Journal	12	Solution paper	Program state-based
S17	[39]	2016	Journal	8	Solution paper	Slicing-based
S18	[40]	2015	Journal	5	Solution paper	Statistical-based
S19	[41]	2015	Conference	5	Solution paper	Spectrum-based
S20	[42]	2015	Conference	6	Solution paper	Spectrum-based
S21	[43]	2016	Conference	8	Experience paper	Statistical-based
S22	[44]	2016	Conference	23	Evaluation paper	Spectrum-based
S23	[45]	2016	Conference	2	Solution paper	Spectrum-based
S24	[46]	2016	Conference	9	Evaluation paper	Spectrum-based
S25	[47]	2017	Conference	2	Evaluation paper	Spectrum-based
S26	[48]	2017	Conference	10	Validation paper	Spectrum-based
S27	[49]	2017	Conference	0	Solution paper	Data mining-based
S28	[50]	2011	Conference	32	Evaluation paper	Spectrum-based
S29	[51]	2009	Symposium	55	Evaluation paper	Spectrum-based
S30	[52]	2009	Conference	241	Solution paper	Hybrid
S31	[9]	2013	Symposium	16	Evaluation paper	Spectrum-based
S32	[53]	2013	Symposium	1	Evaluation paper	Spectrum-based
S33	[54]	2012	Symposium	28	Solution paper	Spectrum-based
S34	[55]	2009	Conference	26	Solution paper	Spectrum-based
S35	[56]	2009	Conference	15	Solution paper	Hybrid
S36	[57]	2017	Symposium	15	Solution paper	Spectrum-based
S37	[58]	2013	Journal	20	Solution paper	Spectrum-based
S38	[59]	2014	Journal	24	Solution paper	Spectrum-based
S39	[60]	2014	Journal	4	Solution paper	Spectrum-based
S40	[61]	2011	Journal	27	Solution paper	Hybrid
S41	[62]	2018	Journal	2	Solution paper	Spectrum-based
S42	[63]	2016	Conference	9	Solution paper	Statistical-based
S43	[64]	2009	Conference	13	Solution paper	Spectrum-based
S44	[65]	2015	Workshop	0	Solution paper	Statistical-based
S45	[66]	2017	Journal	5	Experience paper	Spectrum-based
S46	[67]	2014	Journal	65	Evaluation paper	Spectrum-based
S47	[68]	2018	Journal	1	Solution paper	Spectrum-based
S48	[69]	2018	Journal	1	Solution paper	Spectrum-based
S49	[70]	2018	Journal	1	Solution paper	Misc.
S50	[71]	2017	Journal	5	Solution paper	Statistical-based
S51	[72]	2014	Journal	106	Solution paper	Spectrum-based
S52	[73]	2012	Journal	59	Solution paper	Statistical-based
S53	[7]	2012	Journal	79	Solution paper	Machine learning-based
S54	[74]	2014	Journal	8	Solution paper	Statistical-based
S55	[75]	2017	Journal	10	Solution paper	Statistical-based

3.1. RQ1. What are the selected studies demographics and characteristics?

Out of the 109 studies rigorously investigated based on all the criteria, 55 articles were discarded and 54 were selected as the final studies. The 55 articles were analyzed deeply in order to answer all the RQs outlined in Section 2.1. Table 4 presents the list of the selected studies with details.

3.1.1. Publication over time

Fig. 3 gives the number of papers published with respect to each year from 2009–18. Throughout the years, researchers in the field of SFL have shown some attention in the study of multiple faults. Looking at Fig. 3, 2010 was the less active year where only one paper was published (S5). Therefore, all the high-ranking CORE Conferences, Workshops, and Symposiums have not published any work related to MFL this year. The single article published in 2010 (S5) was presented in the International Conference on Predictive Modes in Software Engineering (PROMISE). However, the spike of other sources from 2013 to 2018 is explained by the fact that on the previous years (2009 and 2011), there were articles published to study the effect of multiple faults in a program under test with respect to the faults interaction and the effect it has on localization effectiveness [27, 50, 51]. Hence, these years also witnessed the publication of respective articles that reignite the interest in MFL research [7, 54, 61]. The reader will also observe that in 2017 and 2018, there are few papers published in comparison to 2016. That could be explained by the fact that some of the most popular Conferences, Symposiums, and Journals have not produced papers in these years. Conferences and Symposiums like International Symposium on Software Reliability Engineering (ISSRE) and International Conference on Automated Software Engineering (ASE) have no papers in these years

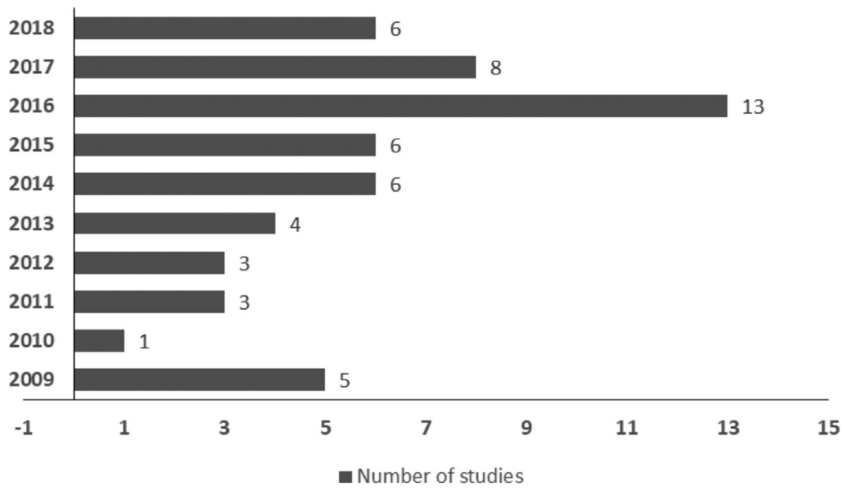


Fig. 3. Number of articles published per year.

(2017 and 2018). In totality, despite some decrease in publication in some years, the research activity on MFL continue to gain momentum with stable growth, particularly during the last 5 years (2014 to 2018).

3.1.2. *Publication channel and quality scores*

Table 4 list the publication channels, publication year, citation count, research type, and SFL technique for each study. Overall, four distinct publication channels were identified (i.e. Journal, Conference, Symposium, and Workshop). Most of the studies were published in Journals, where 28 studies (50.91%) of the selected studies appeared in Journals, 20 studies (36.36%) appeared in Conference, 6 studies (10.91%) were published in Symposium, and lastly, 1 study (1.82%) was presented in Workshop proceedings. This also shows the general quality of the selected studies, because 50% of the studies were published in Journals.

For the research type, the classification is driven from our previous systematic mapping study [17] where studies were classified based on the type of research that was conducted. From our selected studies in this paper, we identified four research types which are solution paper, evaluation paper, validation paper, and experience paper. Therefore, 74.54% of the studies are solution papers, 20% are evaluation papers, and 3.64% are experience papers, followed by validation papers with 1.82%. This is not surprising due to the fact that in this research domain, researchers are always proposing solutions to address critical issues. Hence, one can say that out of the 55 selected studies, 74.54% of them are focused on proposing solutions in localizing multiple faults effectively. With respect to SFL techniques, it is not surprising that spectrum-based fault localization technique (SBFL) is the most utilized, whereby 61.11% of the selected studies are based on it. This is not new information where previous review papers in the field of SFL have shown the same trend [10, 17]. We further assessed the studies for quality based on our quality criteria in Section 2.6. The first author coordinated the quality evaluation extraction process, the first author assessed every paper and allocated the papers to each of the other authors of this study to assess independently. When there was a disagreement, we discussed the issues until we reached agreement. Hence, when a criteria is scored, all the authors are console to give their final verdict on the score so as to get an appropriate and an unbiased score. The score for each study is shown in Table 5. The results of the quality analysis show that all studies score more than 2. Nine studies score 7 (S7, S10, S11, S39, S40, S49, S50, S51, and S52) and five studies score 6 (S3, S16, S30, S36, and S37).

Hence, we identified eight articles (S1, S23, S25, S27, S32, S43, S44, and S48) that did not clearly state their contributions and limitations. We found out that 17 (31.48%) articles were judged to have at least minimal amount of biases.

Furthermore, the quality score for the studies was further investigated in relation to the year the articles were published. Table 6 shows the average quality scores for studies with respect to year of publication. The average quality scores look to be stable and on the increase. The average quality of studies in 2013 is the lowest at 3.87. However, in 2018, the quality has increased significantly. This quality assessment may aid scholars in this research domain to identify relevant papers based on the criteria defined in Section 2.6.

3.1.3. *Publication source*

In relation to publication sources, Table 7 categorizes all the studies with respect to their publication sources. This categorization will help in identifying the publication sources that produce more articles in the study of MFL for the last decade. Furthermore, the publishers of each publication source are also depicted. Overall, we identified 33 sources that published the selected studies. The Journal of Systems and Software, IEEE International Symposium on Software Reliability Engineering (ISSRE), and ACM International Conference on Automated Software Engineering (ASE) were the top contributors with 9, 4, and 3 publications, respectively. We further observed that most of the studies published in the top publication sources (i.e. S2, S10, S11, S17, S39, and S40) have a high-quality score of 6.5 and above from the quality assessment done in Table 5. From Table 7, we identified six publishers which are IEEE with 17 publication sources (17), followed by Springer (7), ACM (3), Wiley (2), Web of Science (WOS) (2), and lastly, Elsevier (2).

3.2. *RQ2. What are the debugging approaches widely used to localize multiple faults?*

This section provides an overview of the debugging approaches used in localizing multiple faults in the literature. Based on our analysis of the selected studies of this SLR, we identified three prominent multiple faults debugging approaches, which are One-bug-at-a-time (OBA) debugging approach, parallel debugging approach, and lastly multiple-bug-at-a-time (MBA) debugging approach as highlighted in Fig. 4 and Table 8. From Table 8, 72.73% of the studies utilized OBA debugging approach, 20% uses parallel debugging approach, and 7.27% of the studies used the MBA debugging approach. In the following subsections, the details of the studies utilizing each of these multiple faults debugging approaches are presented.

3.2.1. *One-bug-at-a-time (OBA) debugging approach*

OBA debugging approach is one of the most popular multiple faults debugging approach in the research domain as shown in Table 8. For

Table 5
Quality evaluation of the selected studies.

Study	QC1	QC2	QC3	QC4	QC5	QC6	Quality total score
S1	1	0.5	0.5	0	0	0	2
S2	1	1	1	1	1	1.5	6.5
S3	0.5	1	1	1	1	1.5	6
S4	0.5	1	1	1	0	1.5	5
S5	0.5	1	0.5	1	1	0	4
S6	1	0.5	0.5	1	0	0	3
S7	1	1	1	1	1	2	7
S8	1	1	1	1	1	1.5	6.5
S9	0.5	1	1	1	1	2	6.5
S10	1	1	1	1	1	2	7
S11	1	1	1	1	1	2	7
S12	0.5	1	0.5	1	0	1	4
S13	0.5	0.5	1	1	0	1	4
S14	0.5	1	1	1	0	2	5.5
S15	1	1	0.5	1	1	0	4.5
S16	0.5	1	1	1	1	1.5	6
S17	0.5	1	1	1	1	2	6.5
S18	0.5	1	1	1	1	2	6.5
S19	1	0.5	0.5	1	0	1.5	4.5
S20	1	0.5	0.5	1	0	1	4
S21	1	0.5	1	1	1	0	4.5
S22	0.5	0.5	0.5	1	1	0	3.5
S23	0.5	0.5	0.5	0	0	0	1.5
S24	0.5	0	1	1	1	0.5	4
S25	1	1	0.5	0	0	0	2.5
S26	0.5	0.5	1	1	1	0.5	4.5
S27	1	0.5	0.5	0	0	0	2
S28	0.5	1	1	1	0	1.5	5
S29	0.5	1	1	1	0	1.5	5
S30	1	0.5	1	1	1	1.5	6
S31	0.5	1	1	1	0	1.5	5
S32	1	0.5	1	0	0	0	2.5
S33	1	0.5	0.5	1	1	1.5	5.5
S34	1	0.5	0.5	1	0	1.5	4.5
S35	1	0.5	1	1	1	1	5.5
S36	0.5	1	1	1	1	1.5	6
S37	0.5	1	1	1	1	1.5	6
S38	0.5	1	1	1	1	2	6.5
S39	1	1	1	1	1	2	7
S40	1	1	1	1	1	2	7
S41	0.5	1	1	1	1	1	5.5
S42	0.5	0.5	0.5	1	1	0	3.5
S43	1	0	0.5	0	0	1.5	3
S44	1	0.5	0.5	0	1	0	3
S45	0.5	1	1	1	1	1	5.5
S46	0.5	1	1	1	1	1	5.5
S47	0.5	1	1	1	1	2	6.5
S48	0.5	1	1	0	1	2	5.5
S49	1	1	1	1	1	2	7
S50	1	1	1	1	1	2	7
S51	1	1	1	1	1	2	7
S52	1	1	1	1	1	2	7
S53	0.5	1	1	1	1	2	6.5
S54	1	1	1	1	1	1.5	6.5
S55	0.5	1	1	0	0	1.5	4

Table 6
Average quality scores for studies by publication date.

	Year									
	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018
Number of studies	5	1	3	4	4	6	6	13	7	6
Mean quality score	4.8	4	5.67	6.33	3.87	6.5	5.25	4.39	4.92	6.42
Standard deviation of quality score	1.41	0	2.11	1.49	3.19	1.22	3.48	5.21	4.43	2.09

OBA, a developer is task to neutralize a single fault per debugging iteration. In a scenario where a program has two faults, this means we need two debugging iterations to neutralize the two faults in the program under test. This process is performed iteratively until all the faults are found and fixed. Various fault localization techniques such as SBFL technique, statistical-based technique, and machine learning techniques have utilized the OBA approach in localizing multiple faults [5, 51, 72, 73, 76].

To further explain these approaches (OBA, parallel debugging, and MBA approach), a running example is given in Fig. 5. The figure gives a program P with a test suite *T*. Test suite *T* is composed of 10 test cases (t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9, t_{10}) with 13 program statements where

Table 7
Publication Source.

Publication source	Publishers	Studies	No.
Journal of Systems and Software	Elsevier	S9, S39, S40, S38, S18, S17, S14, S11, S10	9
International Symposium on Software Reliability Engineering (ISSRE)	IEEE	S2, S32, S33, S29	4
International Conference on Automated Software Engineering (ASE)	ACM	S3, S30, S43	3
Information and Software Technology	Elsevier	S37, S16	2
Transactions on Software Engineering	IEEE	S50, S47	2
IEEE Access	IEEE	S48, S49	2
Software: Practice and Experience	Wiley	S45, S41	2
IEEE Transactions on Reliability	IEEE	S51, S53	2
International Conference on Software Maintenance (ICSM)	IEEE	S28, S34	2
International Conference on Software Testing, Verification, and Validation (ICST)	IEEE	S24, S26	2
International Symposium on Software Testing and Analysis (ISSTA)	ACM	S36, S4	2
Software Quality Journal	Springer	S54, S55	2
IEEE Transaction on Systems, Man, and Cybernetics	IEEE	S52	1
Computing and Informatics	WOS	S13	1
Frontier of Computer Science	Springer	S12	1
Journal of Software: Evolution and Process	Wiley	S46	1
Journal of Information Processing	WOS	S15	1
Empirical Software Engineering	Springer	S7	1
Automated Software Engineering	Springer	S8	1
International Conference on Software Engineering (ICSE)	IEEE	S19	1
Australasian Software Engineering Conference (ASWEC)	IEEE	S20	1
International Conference on Software Analysis, Evolution, and Reengineering (SANER)	IEEE	S21	1
International Conference on Software Maintenance and Evolution (ICSME)	IEEE	S22	1
International Conference on Software Analysis, Testing and Evolution	IEEE	S23	1
International Conference on Software Quality, Reliability and Security	IEEE	S25	1
International Conference on Tools with Artificial Intelligence	IEEE	S27	1
International Symposium on Empirical Software Engineering and Measurement (ESEM)	IEEE	S31	1
International Conference on Quality Software (QSIC)	IEEE	S35	1
Hardware and Software: Verification and Testing (HVC)	Springer	S42	1
International Workshop on Structured Object-oriented Formal Language and Method	Springer	S44	1
International Conference on Software Security and Reliability Companion	IEEE	S1	1
International Conference on Predictive Modes in Software Engineering (PROMISE)	ACM	S5	1
International Conference on Software Engineering, Research and Practice (SERP)	Springer	S6	1

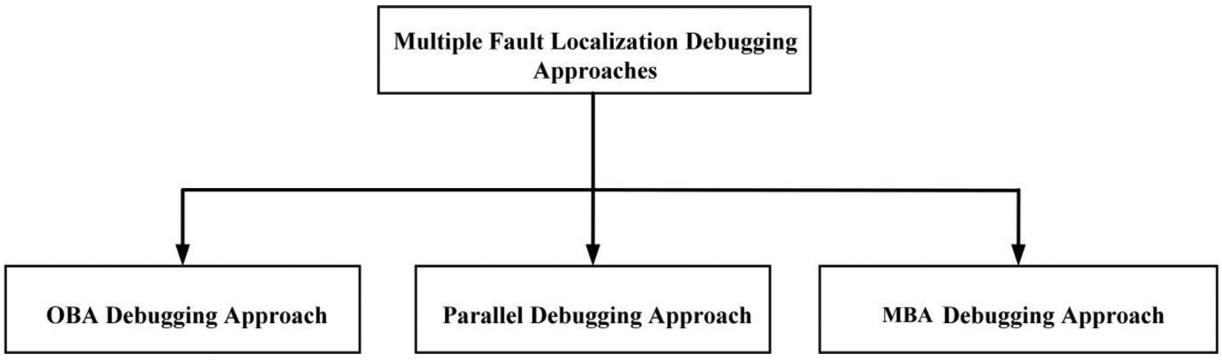

Multiple Fault Localization Debugging Approaches
OBA Debugging Approach
Parallel Debugging Approach
MBA Debugging Approach

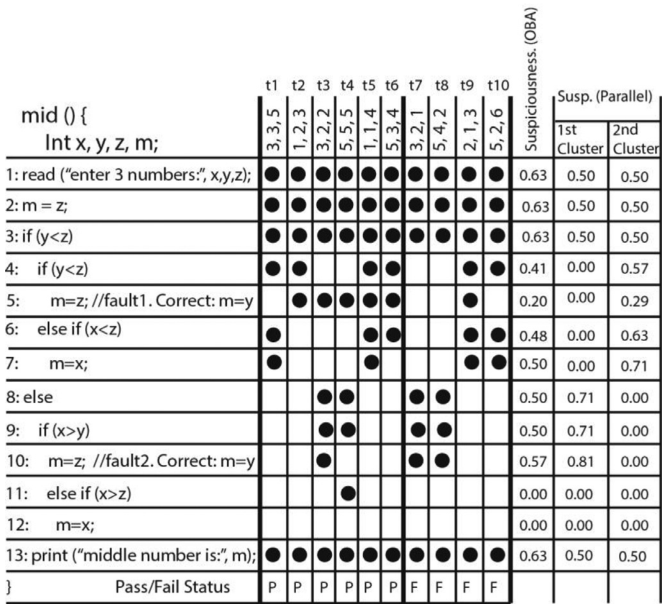
Fig. 4. Multiple fault localization debugging approaches.

Table 8
Multiple fault localization debugging approaches.

Debugging approaches	Studies	No.	%
One-bug-at-a-time (OBA) debugging approach	S3, S4, S5, S6, S7, S8, S9, S11, S13, S14, S16, S17, S18, S19, S20, S21, S22, S23, S24, S25, S26, S27, S28, S29, S30, S31, S36, S37, S38, S41, S42, S43, S44, S45, S46, S47, S48, S51, S52, S55	40	72.73
Parallel debugging approach	S1, S2, S12, S15, S32, S33, S34, S39, S50, S53, S54	11	20
Multiple-bug-at-a-time (MBA) debugging approach	S10, S35, S40, S49	4	7.27

statement 5 and statement 10 are both faulty. A statement execution labeled with "●" signifies that the statement is executed by the test case in that test run, and empty otherwise. Test cases (t_7, t_8, t_9, t_{10}) are failed test cases while (t_1, t_2, t_3, t_4, t_5, t_6) are passed test cases. In this example, the suspiciousness of each statement is computed using Ochiai coefficient with two results given, one for OBA and one for Parallel debugging.

For OBA approach is simple, looking at the suspiciousness scores of the statements, the developers will clearly identify a faulty statement 10 (with suspicious value of 0.57) by checking five statements. Hence,



mid () {	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	Suspiciousness. (OBA)	Susp. (Parallel)	
Int x, y, z, m;	3, 3, 5	1, 2, 3	3, 2, 2	5, 5, 5	1, 1, 4	5, 3, 4	3, 2, 1	5, 4, 2	2, 1, 3	5, 2, 6		1st Cluster	2nd Cluster
1: read ("enter 3 numbers:", x,y,z);	●	●	●	●	●	●	●	●	●	●	0.63	0.50	0.50
2: m = z;	●	●	●	●	●	●	●	●	●	●	0.63	0.50	0.50
3: if (y<z)	●	●	●	●	●	●	●	●	●	●	0.63	0.50	0.50
4: if (y<z)	●	●			●	●			●	●	0.41	0.00	0.57
5: m=z; //fault1. Correct: m=y		●	●	●	●	●			●		0.20	0.00	0.29
6: else if (x<z)	●				●	●			●	●	0.48	0.00	0.63
7: m=x;	●				●				●	●	0.50	0.00	0.71
8: else			●	●			●	●			0.50	0.71	0.00
9: if (x>y)			●	●			●	●			0.50	0.71	0.00
10: m=z; //fault2. Correct: m=y			●				●	●			0.57	0.81	0.00
11: else if (x>z)				●							0.00	0.00	0.00
12: m=x;											0.00	0.00	0.00
13: print ("middle number is:", m);	●	●	●	●	●	●	●	●	●	●	0.63	0.50	0.50
} Pass/Fail Status	P	P	P	P	P	P	F	F	F	F			

Fig. 5. Running program with two faults.

based on OBA, the fault found in statement 10 has to be fixed and the program has to be re-tested to find the second fault (at statement 5). This procedure is distinct to the one for parallel debugging and MBA debugging approaches. For parallel debugging approach, the suspicious score result was split into two clusters which represents the two *fault-focused* clusters generated based on the failed tests execution profiles similarity as done in a previous study [78]. One can see that the first *fault-focused* cluster (given to the 1st developer) targets the fault at statement 10 (with suspicious score of 0.81), while the second *fault-focused* cluster targets the fault at statement 5 (with suspicious score of 0.29). Hence, both faults can be localized simultaneously in parallel using two developers in this scenario. For MBA debugging approach, looking at the figure (Fig. 5), the statements suspicious scores for MBA debugging approach was not given. This is because the suspicious score for OBA can be used to illustrate how to localize the two faults in the program using the MBA debugging approach. Hence, based on MBA debugging approach, the developer will not stop after locating the first fault in statement 10. Therefore, the fault searching exercise will continue to the pre-specified search cap (a limited statement search space) set by the developer. The approach of setting a cap (i.e. a developer can search 70% or 60% of the suspicious statement ranking list before moving to the next iteration) has been adopted by existing studies adopting MBA debugging approach [4, 32]. In this debugging approach (MBA debugging approach), the researchers normally put a fault search cap to limit the fault search space in the fault ranking list generated. When the cap is reached and all the faults are not found in a given iteration, the existing identified faults will be fixed and a further retest (moving to next iteration) will be conducted to identify the rest of the faults.

Moreover, by utilizing OBA debugging approach, additional effort is needed for the developer to find and fix the faults. In the process, more faults can also be created during regression testing, while also resulting in longer time-to-delivery of software programs [2, 55]. We observed that out of the 40 studies that adopted the OBA debugging approach, 11 studies (S3, S6, S8, S13, S14, S17, S21, S22, S23, S24, and S42) were published in 2016 which was the most active year for studies adopting it. Followed by 2015 and 2017 with 6 each, respectively. Subsequently, studies that utilized the OBA debugging approach will be presented. Abreu et al. presented an approach named BARINEL, which combines the best of SBFL techniques and MBD techniques [52]. In this approach (BARINEL), a program was modeled with execution traces while Bayesian reasoning was used to deduce multiple fault candidates. The approach showed some promise but was found to be more effective in the context of a single fault. In order to localize multiple faults effectively, Dean et al. proposed a new algorithm based on linear programming model which is design to be specifically effective in the context of multiple faults. The result shows that the proposed algorithm outperforms the fault localization techniques compared with [64]. Also, Abreu et al. proposed a new reasoning-based fault localization approach named Zoltar-C. The approach exploits the execution frequency of program components and utilizes Bayesian approach to compute the probabilities of program components containing faults [28]. The result shows that even though the approach performs better, but the concept of program components frequency exploitation does not improve diagnostic accuracy. A crosstab-based statistical approach was proposed by Wong et al. to localize faults using the OBA debugging approach [73]. The experimental results showed that the OBA debugging approach is not as effective. In addition, Xu et al. proposed a fault localization framework that reduces the noise between faults in the presence of multiple faults [58]. The framework uses a chain of key basic blocks of a program and a noise reduction method to improve similarity coefficient metrics.

Furthermore, Wong et al. proposed a modified form of *Kulczynski* similarity coefficient named DStar [72]. Using this coefficient, the higher the computed DStar value, the more effective the technique. The coefficient was tested on multiple-fault programs using the OBA debugging approach and the findings revealed that the technique can localize at least a single fault with high effectiveness. In a study by Perez et al., a new approach named dynamic code coverage (DCC) was proposed [59]. The approach is aimed at reducing the instrumentation overhead of SBFL techniques by mean of using coarser instrumentation. An empirical evaluation on real-world multiple-fault programs shows that the proposed approach can considerably reduce the execution overhead in SBFL techniques. Lucia et al. conducted an investigative study on 40 SBFL techniques in terms of fault localization effectiveness. Their evaluation on both single-fault and multiple-fault programs with the utilization of OBA debugging approach shows that there is no single best SBFL technique in all cases [67]. Hence, the authors also observed that the accuracy of the techniques in localizing multiple faults is lower in comparison to their performance in localizing single faults. Wang and Liu proposed a hierarchical multiple predicate switching technique named HMPS to localize faults effectively [40]. The result indicates that HMPS performs better in comparison to other fault localization techniques (such as Barinel and Ochiai) in identifying the location of faults effectively. In another study by Yu et al., the authors proposed a technique that can be utilized to differentiate failed test cases that execute a single fault from failed test cases that executes multiple faults. Their experiment on multiple-fault subject programs shows that the technique performance is promising [41].

Moreover, a multiple fault localization method was proposed based on Simulink model [43]. The approach used a supervised learning technique named decision tree to cluster failed executions that were likely to have been caused by a single fault. In this process, a rank list based on a statistical debugging technique is generated and developers can use this list to find a fault, fix it, and re-test the Simulink model to localize the remaining faults. Although this approach uses failure clustering method as the basis of classifying failures, it is still based on the OBA debugging approach as indicated in the study. In another study, Lee et al. proposed a weighting technique to improve the effectiveness of SBFL techniques [29]. A weighting value is assigned to test cases that are caused by both single fault and multiple faults. This weighting is primarily done by utilizing information extracted from failed test cases that are caused by multiple faults. The study concluded that weighting failed test cases caused by multiple faults improve the effectiveness of fault localization techniques. Wang et al. proposed a novel fault localization approach based on disparities of dynamic invariants, named FDDI [36]. FDDI selects a highly-suspected function and then applies invariant detection tools to this function separately. Variables that are not in a set of passed/failed test cases indicated by using these tools are picked by FDDI for further analysis. However, in the context of multiple faults, the authors considered failed test cases that execute all faulty statements. Therefore, using the OBA debugging approach, the researchers neutralized single fault each at each debugging iteration to produce a failure-free program.

Furthermore, a weighting technique was proposed by Neelofar et al. using both dynamic program analysis and static program analysis to categorize program statements and rank them based on their likelihood of containing faults. The technique was tested on both single-fault programs and multiple-fault programs. The proposed technique is said to improve the performance of various fault localization metrics up to 20% on single-fault datasets and up to 42% on multiple-fault datasets [66]. A diversity maximization speedup (DMS) strategy was proposed by Xia et al. to aid developers in the test case selection during fault localization to also reduce associated costs [30]. This strategy also helps in targeting critical test cases that are needed to speed up the localization process. The result of the study shows that DMS can aid the existing fault localization techniques in reducing debugging cost of locating multiple faults. A hybrid method was proposed named Stat-slice to locate faults in programs with a large number of faults using the OBA debugging approach [35]. The result shows that the method has considerably reduced fault localization effort. Also, bounded debugging via multiple predicate switching (BMPS) technique based on the OBA debugging approach was proposed [65], and experimentation revealed that multiple faults were localized. Xiaobo et al. conducted an empirical study to explore failure behavior of multiple faults in a program through empirical investigation on real-life systems (Chinese Railway System) [47]. The study showed that unpredictable failure caused by multiple faults is mainly accounted by the interaction of dominant faults during program execution.

Laghari et al. proposed a variant spectrum-based fault localization technique named patterned spectrum analysis [26]. This technique leverages patterns of method calls that are extracted through frequent itemset mining. The result of the study shows that the proposed technique is effective in comparison with normal spectrum-based fault localization techniques. In another work by Zhang et al., the authors conducted an investigation on the impact of cloning failed test cases on spectrum-based fault localization (SBFL) techniques effectiveness. The results on multiple-fault subjects' shows that the fault localization accuracy can be improved, specifically when failed test cases are cloned [33]. Dandon et al. proposed a state dependency probabilistic model and a fault localization approach to aid in describing the control flow dependence between statements in a particular state and to locate faults in both passed and failed test executions [38]. By utilizing the OBA debugging approach, the result shows that the proposed approach outperforms other techniques in locating faults effectively.

Zhang and Santelices proposed a novel static slicing technique named PRIOSLICE. The technique computes a probabilistic model of the dependencies in a program. With this, PRIOSLICE traverses the program backward in an order defined by the computed dependence probabilities to locate faults. The result shows that the technique aid in localizing faults effectively in comparison to the existing static-slicing approaches [39]. Furthermore, an investigative study was conducted by Xia et al. to understand whether SBFL techniques can help developers to improve debugging efficiency in large software projects. The authors conclude that SBFL can aid developers in saving debugging time significantly with good effectiveness [44]. Naish et al. proposed the use of genetic programming in localizing multiple faults. The authors proposed a class of hyperbolic metrics that have a small number of numeric parameters which can be adjusted based on behavioral changes. The evaluation on multiple fault programs shows that the proposed technique outperforms existing metrics that was compared with [42, 62].

In another study that adopts the OBA debugging approach [45]. The authors proposed a test case prioritization algorithm for fault localization. The result shows that the proposed algorithm can aid in reducing debugging effort. In order to clearly understand the properties of effective SFL metrics, Sun and Podgurski conducted an investigative study on various statistical fault localization metrics that have demonstrated good performance prior to their study [46]. The results indicate that metrics that performed better have two characteristics. First, a metric that estimates the failure-causing effect of a program element and secondly, a metric that weights the first element (elements with estimated failure-causing effect) based on the evidence for the existence of faults in other program elements. An iterative user-driven fault localization technique named Swift was proposed by Li et al. [63]. The technique leverages statistical fault localization to identify suspicious program methods, generates high-level queries on the correctness of executions for the most suspicious methods, and lastly uses developers' feedback to improve the localization results generated. Based on empirical evaluation, the result shows some promise. In a study by Perez et al., the authors proposed an approach that can find fault-fixes and categorize them in accordance to the number of faults they fix, to assess the prevalence of single-fault fixes [48]. The result shows that there is a prevalence of single-fault fixes with over 82% of fixes eliminating a single fault from the system. Hence, the authors outline some practical implications of their findings. By adopting the OBA debugging approach, a study by Aribi et al. move to address the issues of multiple faults (which raises various issues due to the complex dependencies between faults and failures) by proposing a new constraint programming model to help speed-up and improve the localization of multiple faults. The result shows some significant improvements [49].

In an effort to enhance existing SBFL techniques, Zhang et al. proposed a technique named PRFL. This technique improves SBFL techniques by differentiating test cases using PageRank algorithm [57]. The result shows some promise when different test cases are considered during fault localization. To further improve SBFL techniques effectiveness to rank faulty program statements to the top of the ranking list effectively, Wang et al. proposed a new technique named SBFL technique via enlarging the non-faulty region to improve fault ranking [69]. Their experimental result on various multiple fault subject programs shows some significant improvements. Zhang et al. who also used the OBA debugging approach proposed a test classification approach to aid in the utilization of unlabelled test cases in localizing faults [31]. There experiment on programs' multiple-fault versions shows that there proposed approach is useful in improving the effectiveness in fault localization. A fault localization approach named BEN was proposed by Ghandehari et al. [68]. The approach was evaluated on both single-fault and multiple-fault programs. The result shows that the approach can identify the location of faults effectively.

Out of the 39 recognized studies that utilized the OBA debugging approach, we have identified five studies that investigate the effect of multiple faults in a program in relation to fault localization effectiveness on both C and Java subject programs [2, 9, 27, 50, 51]. These studies which were conducted between the years (2009–15) were instrumental in understanding faults behavior and the impact of multiple faults on fault localization effectiveness. All of the five studies utilized OBA debugging approach for their experiments.

3.2.2. *Parallel debugging approach*

Parallelization or parallel debugging approach is basically dividing the debugging task into small units so as to allow multiple developers to work on different units [77]. This approach is utilized when a program has multiple faults, mainly to facilitate the debugging process and reduce software time-to-delivery. Failed test cases are clustered and each cluster is combined with all the available passed test cases to create a single *fault-focused* cluster, with the assumption that each *fault-focused* cluster targets a single fault. The *fault-focused* clusters which composed of both failed and passed test cases will be given to separate individual developers to debug in parallel. This debugging approach is different from the OBA debugging approach because a developer does not have to neutralize the faults in many debugging iterations as the case in OBA. We identified 11 studies that adopted the parallel debugging approach (as shown in Table 8). However, out of these studies, 2014 was the most active year where most of these papers were published with three studies (S2, S39, and S54), respectively. Followed by 2016, 2013, and 2012 with two studies each, respectively.

Jones et al. introduced the idea of debugging in parallel by clustering the failed test cases and combining each cluster with all available passed test cases to form a *fault-focused* cluster [77]. These *fault-focused* clusters are then given to developers to debug the faults in parallel. Hence, the authors make a presumption that each *fault-focused* cluster represents a single fault. A recent work by Wolfgang et al. [25] concluded that the presumption does not seem realistic. One of the issues with parallelization is that a single developer can finish a debugging task while other developers are still debugging. Consequently, this can create more faults to the program as fixes given to the first developer who finishes debugging early will probably affect the debugging effort of other developers. Jeffrey et al. proposed a fault localization method based on value replacement to efficiently localize multiple faults. The technique reduces the total time required to locate multiple faults on the order of minutes. Initially, finding and fixing the faults based on OBA debugging approach was considered, before the authors eventually concluded that it would be costly and that such approach also increases time-to-delivery of the software program [55]. As a result, an iterative process was considered, where the technique can iteratively compute a ranking list of program statements with the aim of each ranking list to guide the developer towards faults as quickly as possible. The method understands potential faulty statements based on the occurrence of interesting value mapping pair (IVMP) [55]. It was identified that IVMP normally occurs at faulty statements. It can also occur in statements that are directly related to faulty statements through a dependency edge. The approach performs localization on individual execution iterations to find and fix faults, which have similarities with the approach by Jones et al. [77]. Moreover, it is not guaranteed that each iteration belongs to a single fault. One of the problems with this approach as highlighted by the authors is the high computational requirements. Even though some faults can be identified in a matter of minutes, others can take hours.

In another study by Wei and Han, a parameter-based combination approach (PBC) was proposed to aid in the efficient localization of multiple faults [24]. In the study, a bisection method was utilized for clustering failed test cases to create *fault-focused* clusters while crosstab-based fault localization technique was used for fault localization. The researchers conclude that PBC performs better than Tarantula (OBA debugging approach). A fault localization technique based on radial basis function (RBF) neural network was proposed by Wong et al. [7]. The neural network is trained to study the relationship between test case statements coverage and its corresponding execution results (success or failure). By adopting the parallel debugging approach, the result shows that the proposed technique performs better than several other popular fault localization techniques compared with. In order to improve SBFL techniques effectiveness on multiple faults, Steimann and Frenkel proposed the use of partitioning procedures from integer linear programming [54]. The method will aid in generating various partitions which break down the localization problems into smaller ones. Hence, each partition can be addressed independently by developers. The result shows that the method is useful in localizing multiple faults.

In a recent study by Lamraoui and Nakajima [37], a formula-based approach was proposed consisting of a full flow-sensitive trace formula to localize faults in programs containing multiple faults. The approach combines satisfiability-based (SAT) formula verification techniques and model-based diagnosis theory. It was able to localize the root causes of multiple faults in a program. However, this approach was examined in a relatively small program of the Siemens test suite (Tcas), and therefore, the results cannot be fully generalized. Also, the approach's exclusive utilization of failed test cases alone for localization might be an issue for a large program with several faults where many statements containing faults can be executed by passed test cases which are more common in multiple-fault scenarios [2, 9, 50]. Sun et al. proposed an iterative process for selecting test cases for effective fault localization [34]. This approach works based on the concept lattice of program spectrum method, and in order to localize multiple faults, program statements are classified into three parts, namely dangerous, sensitive, and safe statements. By doing so, developers start by checking the statements that are categorized as dangerous first due to their high probability of containing faults, followed by the rest of the classified statements. A recent study by Gao and Wong proposed a novel approach for localizing multiple faults in parallel [71]. The authors proposed an improved *k*-medoids clustering algorithm to aid in the effective identification of the relationship between failed test cases and their corresponding faults. The study concludes that their proposed approach performs better in terms of efficiency and effectiveness in comparison to other debugging approaches. Huang et al. conducted an empirical study on fault isolation in fault localization [53]. The authors analyzed the effectiveness of six fault localization techniques and two clustering algorithms (k-means and hierarchical clustering) for fault isolation. The result shows that k-means outperformed hierarchical clustering for isolating faults in fault localization.

Wang et al. proposed a weighted attribute-based strategy (WAS) for cluster test selection. WAS conduct clustering on tests using weighted execution profiles by also considering the suspiciousness of program statements. The evaluation on various multiple-fault programs shows that WAS outperforms other clustering strategies for tests clustering in fault localization [60]. A more recent approach proposed named Hierarchy-Debug [74] that aims to localize latent bugs. In this approach, a hierarchical clustering algorithm is applied to cluster predicates to support scalability in localizing multiple bugs. The results showed that the approach can aid developers in grouping predicates caused by multiple bugs.

3.2.3. *Multiple-bug-at-a-time (MBA) debugging approach*

MBA approach to debugging multiple faults is an appealing concept and also effective when done the right way. In this paper, we define MBA debugging approach as the art of localizing most if not all faults in a single debugging iteration, with respect to the fault search cap set by the developer. This sounds ambitious, but it is a possible way to improve fault detection rate and reduce debugging time which will directly improve the time-to-delivery of software systems. Another obvious advantage of the MBA debugging approach is the reduction of computation overhead of utilizing clustering algorithms in identifying failure-to-fault relationships as in the case with parallel debugging approach. However, this approach (MBA debugging approach) has enjoyed little attention in recent years with only four studies utilizing it in the last decade (see Table 8). Hence, out of these studies, 2018 was the most active year where most of these papers were published with two studies (S10 and S49) respectively. An earlier study by Gong et al. has made a suggestion on localizing more than one fault in the first debugging iteration, in extension, debugging all the faults simultaneously [79]. However, their proposal has not been implemented or empirically validated.

Abreu et al. proposed a reasoning fault localization approach named Zoltar-M. the approach generates multiple-fault diagnoses and rank them in order of their probability [56, 61]. The authors aim is to effectively localize multiple faults simultaneously. The results show that the approach outperforms various fault localization techniques. A recent study by Zheng et al. has also put an effort to localize faults simultaneously by proposing a Fast Software Multi-Fault Localization Framework based on Genetic Algorithms (FSMFL) [32]. FSMFL localize multiple faults simultaneously based on genetic algorithm with simulated annealing. The experimental result shows that FSMFL is competitive in single fault localization and superior in multiple fault localization. Zakari et al. proposed a fault localization technique based on complex network theory named FLCN. The technique is aimed to improve localization effectiveness in programs with single fault and multiple faults and to aid developers to localize multiple faults simultaneously in a single debugging iteration. The experimental result shows that FLCN performs better in comparison with the existing fault localization techniques [70].

3.3. RQ3. What are the datasets, datasets programming languages, and fault types used in localizing multiple faults?

In answering this research question, we observed that the seven programs from the Siemens test suite dataset (i.e. tcas, print_tokens, print_tokens2, schedule, schedule2, replace, and tot_info) were the most utilized with 33 studies adopting them for their empirical experiments, followed by UNIX programs (i.e. gzip, grep, make, flex, and sed) with 26 studies adopting them. As highlighted in Table 8, in most cases, these programs are used together with other subject programs in the studies (i.e. S4, S5, S8, S55, and S10). In other words, some of the studies used more than one dataset and programs for their experiments.

Defect4j dataset is recently considered as a good dataset for MFL, which composed of programs that are larger in size with real faults [17]. However, we observed that this dataset is used by only five studies out of the 55 selected studies (S3, S10, S22, S36, and S42). Furthermore, the identification of datasets programming languages is vital in knowing the diversity of programs used by studies. From our selected studies, we identified three major programs programming languages that were used, which are C, C++, and Java. 33 studies used C, 10 studies used Java, 9 studies used programs with both C and Java languages, and 1 study uses programs with both C and C++ languages. In total, one study uses programs with the combination of five languages which are Java, JavaScript, Python, Ruby, and Scala [48] with another study that did not specify the dataset programming language they used [43].

The excessive utilization of programs from Siemens test suite infers that most of the multiple faults used for the experiments of those respective studies are seeded faults or sometimes called artificial faults. These types of faults are generally seeded manually into program versions to create program versions containing many faults [2] or sometimes using mutation fault injection techniques [71, 81, 82]. This, however, shows that most of the studies have more than a decent level of biases in them because even though earlier studies argued that mutation-based faults can be useful to represent real faults and provide reliable results in program debugging experiments, recent studies have said otherwise [83]. Having said that, from Table 8, we identified 39 (70.91%) studies that strictly utilized seeded faults, 10 (18.18%) studies utilized real faults, and lastly, 6 (10.91%) studies utilize both real and seeded faults. Looking at the result, despite the clear importance of using real faults (in terms of enhancing result generalization and the reduction of bias to meet up with the industry standard) for program debugging experiments in the context of multiple faults, more than 70.91% of the selected studies still used mutation-based faults (seeded faults).

Table 8a

However, despite this trend, we observed that 70% of the studies that used real faults are published between 2016–17 (S3, S21, S22, S24, S25, S26, and S36). This indicates the progressive change in the research domain in recent years toward the utilization of real faults. An interesting discovery is that 8 out of the 10 programs that use Java programs utilize real faults for their experiments. This means that most of the existing datasets that have real industrial faults suitable for multiple faults experiments are in Java programming language.

3.4. RQ4. What are the evaluation metrics used in multiple fault localization (MFL)?

To understand the evaluation metrics that are the most utilized by the selected studies, we classify the existing identified evaluation metrics (identified from the selected studies) and categorized the studies based on which metric they used. Table 9 shows the list of evaluation metrics with respect to the studies that utilized them. Overall, we identify 20 evaluation metrics as highlighted in Table 9.

The result of this question reveals that Expense score metric is the most utilized with 15 studies, followed by Exam score and Wasted effort with 14 and 8, respectively. Furthermore, Wilcoxon Signed-rank test (WSR) has become a most when evaluating MFL proposals, which was utilized by some high-quality papers (S10, S20, S26, S50, S51, and S55) with an average quality score of 5.9 (presented in Table 5).

4. Discussion

In this article, we conducted a systematic review (SLR) on MFL (a sub-domain of SFL). SFL has received significant attention from the software engineering research community in the past few decades. Of recent, the issue of MFL a sub-domain in SFL has become a crucial concern in the domain. Localizing multiple faults is difficult which has been receiving quite a reasonable amount of attention in the last 10 years (as shown in Fig. 3). This section summarizes and discusses the results related to the RQs by presenting the research findings, research challenges, and the direction for future work.

4.1. Research findings

In this SLR study, we examined the current knowledge in MFL by selecting 55 studies from a total of 109 (excluding those studies that did not fulfill our IC/EC and QA criteria). These studies were then analyzed and evaluated to aid in answering our defined RQs (Table 1). The findings of our study are as follows.

From the selected studies demographics point of view, with respect to the publication trend, the result shows some momentum with stable growth specifically in the last 5 years. We observed that 2010 was the least active year with only one study published [28]. We found out that the increase of articles from the year 2013 to 2018 was partially due to the fact that critical studies such as [27, 50, 51] were published prior to these years. These studies primarily study the effect of multiple faults with regards to fault localization effectiveness. Furthermore, 50.91% of the selected studies were published in Journals, which was the highest publication among the publication channels. These shows the general quality of the selected studies used in this paper. Another observation is that 12.96% of the selected studies have a total quality score of 7, while 11.11% has a quality score of 6. This shows that 24.07% of the studies have a total quality score of more than 6. However, looking at the average quality score with respect to the year of publication, we observed that from 2013, there is a steady increase in the quality of papers published on MFL. On the publication source, three sources were identified to be more prominent, which are Journal of Systems and Software, International Symposium on Software Reliability Engineering (ISSRE), and International Conference on Automated Software Engineering (ASE) with 9, 4, and 3 publications, respectively.

In MFL research domain, we found out that a significant number of studies have adopted the OBA debugging approach with 72.73, followed by parallel debugging approach with 20%, and MBA debugging approach with 7.27%. Some critical observations with regard to debugging approaches are, for OBA debugging approach, we have seen more papers published in 2016. In that year (2016), we found out that 28.20% of the studies published adopted OBA debugging approach. For parallel debugging approach and MBA debugging approach, the most active years are 2014 and 2018 with 27.27% and 50% of studies adopting them, respectively.

With respect to RQ3, we observed that programs from the Siemens test suite were the most utilized followed by UNIX programs with 33 and 26 studies utilizing them, respectively. However, Defect4j datasets utilization in the MFL domain is on the rise in the last two years. Furthermore, we found out that datasets that were written in C and Java languages are the most utilized. Out of the selected studies, 60% of the studies used C datasets while 18.18% of the studies used Java datasets. With respect to fault types, we observed that 70.91% of the studies used

Table 8a
Studies and their statistics based on datasets, programming language, fault types, and evaluation metric.

Datasets	Dataset programming language	Fault types	Evaluation metric	Study
Siemens test suite	C	Seeded faults	Exam score	S1
15 Java programs (AC Codec 1.3, AC Lang 3.0, Barbecue Rev. 87, Daikon 4.6.4, Eclipse Draw2d 3.4.2, Eventbus 1.4, HTML Parser 1.6, Jaxen 1.1.5, JDepend 2.9, Jester 1.37b, JExel 1.0.0b13, JParsec 2.0, Mime4j 0.5, Time & Money Rev. 207, XMLSec 3.0)	Java	Real faults	Wasted effort	S2
Defect4j dataset (Math, Lang, Time, Chart, Closure)	Java	Real faults	Wasted effort	S3
UNIX program (Gzip) Siemens test suite (Replace) and Space	C	Seeded faults	Expense score	S4
Siemens test suite, Space, UNIX programs (Gzip and Sed)	C	Seeded and real faults	Wasted effort	S5
Siemens test suite and Space	C	Seeded faults	Expense score	S6
UNIX programs (Flex, Gzip, and Sed) Siemens test suite programs (Replace and Schedule), and Space	C	Seeded faults	Expense score	S7
Siemens test suite, UNIX programs (Sed, Flex, Grep, and Gzip), and Space	C	Seeded faults	Cost	S8
UNIX programs (Flex, Grep, Gzip, and Sed), Siemens test suite and Space	C	Seeded faults	Precision and recall	S9
Siemens test suite, Space, Linux, and Defect4j	C and Java	Seeded and real faults	Exam score, Exam$_F$, Exam$_L$, Top-N, and Wilcoxon Signed-rank test (WSR)	S10
Siemens test suite, Space, and UNIX programs (Flex, Gzip, Grep, and Sed)	C	Seeded faults	Expense score, Exam score	S11
Siemens test suite programs (Schedule1, Schedule2, Print_tokens1, Print_tokens2, Tot_info), NanoXML, XML-Security	C and Java	Seeded and real faults	Expense score	S12
Siemens test suite, and UNIX programs (Gzip, Grep, and Flex)	C	Seeded faults	T-score	S13
UNIX programs (Sed, Gzip, Grep, Make, and Flex) and Space	C	Seeded faults	Exam score	S14
Siemens test suite (Tcas) and Bekkouche's benchmark	C	Seeded faults	Code size reduction (CSR)	S15
Siemens test suite and UNIX programs (Flex, Grep, Gzip, and Sed)	C	Seeded faults	T-score	S16
NanoXML, XML-Security, Jmeter, Jaba, Ant, PDFBox	Java	Seeded faults	Expense score	S17
UNIX programs (Flex, Grep, Gzip, and Sed) and Space	C	Seeded faults	Exam score	S18
UNIX programs (Sed, Flex, Grep, and Gzip) and Space	C	Seeded faults	Precision, recall, and F-measure	S19
Siemens test suite and UNIX programs	C	Seeded faults	Wilcoxon Signed-rank test (WRS) and Expense score	S20
Models from Delphi automotive (MS, MC, MGL models)	Nil	Real faults	Cost	S21
Defect4j	Java	Real faults	Success rate and debugging time	S22
Siemens test suite and UNIX programs (Flex, Grep, Gzip, and Sed)	C	Seeded faults	Expense score	S23
Daikon, Eventbus, Jaxen, Jster, Jexel, Jparsee, AC Codec, AC Lang, EclipseDraw2d, HTML Parser	Java	Real faults	Cost	S24
Driver, Tools, Spcecificlib, and Workspace	C and C++	Real faults	Nil	S25
Dataset by [80]	Java, JavaScript, Python, Ruby, and Scala	Real faults	Effort and Wilcoxon Signed-rank test (WSR)	S26
Siemens test suite	C	Seeded faults	Exam score	S27
UNIX programs (Flex, Gzip, and Sed) Siemens test suite programs (Replace and Schedule), and Space	C	Seeded faults	Nil	S28
Siemens test suite	C	Seeded faults	Nil	S29
Siemens test suite, UNIX programs (Gzip and Sed), and Space	C	Seeded faults	Wasted effort	S30
NanoXML v.1, v.2, Jmeter v.3, Jtopas v.1, Bitset, FilteredRowSet, Math, and RE	Java	Seeded and real faults	Expense score	S31
UNIX programs (Flex and Grep)	C	Seeded faults	Exam score	S32
Jexel, Jester, AC Codec, Jparsec, AC Collections, AC Lang, and Daikon	Java	Seeded faults	Exam score	S33
Siemens test suite	C	Seeded faults	Expense score	S34
Siemens test suite	C	Seeded faults	Wasted effort	S35
Defect4j	Java	Real faults	Wasted effort and Top-N	S36
Jtopas, XMLSecurity, Ant, Jmeter, NanoXML	Java	Real faults	Exam score	S37
NanoXML, Xstream, JGAP, XML-Security, Jmeter	Java	Seeded faults	Expense score	S38
UNIX programs (Flex, Grep, Gzip, Sed), Space, Make, and Ant	C and Java	Seeded faults	Precision and recall	S39
Siemens test suite, UNIX programs (Gzip and Sed), and Space	C	Seeded faults	Wasted effort	S40
UNIX programs (Flex, Grep, Gzip, and Sed)	C	Seeded an real faults	Execution time and expense score	S41
Siemens test suite, Defect4j, and SAEG	C and Java	Seeded and real faults	Expense score	S42
Siemens test suite and Space	C	Seeded faults	Nil	S43
Siemens test suite	C	Seeded faults	Expense score	S44

(*continued on next page*)

Table 8a (*continued*)

Datasets	Dataset programming language	Fault types	Evaluation metric	Study
UNIX programs (Flex and Sed) and Space	C	Seeded faults	Wasted effort	S45
Siemens test suite, space, and NanoXML, NanoXML-Security	C and Java	Real faults	Expense score and proportion of bugs localized	S46
Siemens test suite and UNIX programs (Flex, Grep, Gzip, and Sed)	C	Seeded faults	Exam score and efficiency	S47
Siemens test suite	C	Seeded faults	Improvement metric	S48
Siemens test suite and UNIX programs (Gzip and Sed)	C	Seeded faults	Exam score and incremental developer expense	S49
UNIX programs (Gzip, Grep, Flex), Make, Ant, Socat, and Xmail	C and Java	Seeded faults	Average number of statements examined, T-exam score, Wilcoxon Signed-rank test (WSR), and efficiency	S50
Siemens test suite, UNIX programs (Gzip, Grep, Flex, and Sed), Make, Ant, Space, and Unix suite programs (Cal, Checkeg, Col, Comm, Crypt, Look, Sort, Spline, Tr, and Uniq)	C and Java	Seeded faults	Exam score, cumulative number of statements examined, and Wilcoxon Signed-rank test (WSR)	S51
Siemens test suite, UNIX programs (Grep, Gzip), Make Unix suite programs, and Space	C	Seeded faults	Exam score	S52
Unix suite programs, UNIX programs (Grep), Space, Make, and Ant	C and Java	Seeded faults	Exam score	S53
Siemens test suite, Space, and Bash	C	Seeded faults	T-score	S54
Siemens suite, the Unix suite, gzip, grep, make, sed, and Ant	C and Java	Seeded faults	Exam score, Average number of statements examined, and Wilcoxon Signed-rank test (WSR)	S55

Table 9
Evaluation metrics utilized by the selected studies.

Metric	No.	Studies
Expense score	15	S4, S6, S7, S11, S12, S17, S20, S23, S31, S34, S38, S41, S42, S44, S46
Exam score	15	S1, S10, S11, S14, S18, S27, S32, S33, S37, S47, S49, S51, S52, S53, S55
Wasted effort	8	S2, S3, S5, S30, S35, S36, S40, S45
Wilcoxon Signed-rank test (WSR)	6	S10, S20, S26, S50, S51, S55
T-score	3	S13, S16, S54
Precision and recall	3	S9, S19, S39
Cost	2	S8, S24
Top-N	2	S10, S36
Efficiency	2	S47, S50
Average number of statements examined	2	S50, S55
Code size reduction (CSR)	1	S15
F-measure	1	S19
Success rate and debugging time	1	S22
Effort	1	S26
Execution time	1	S41
Proportion of bugs localized	1	S46
Improvement metric	1	S48
Incremental developer expense	1	S49
Cumulative number of statements examined	1	S51
T-exam score	1	S50

seeded faults, 18.18% of studies used real faults, and 10.91% of the studies utilize both real and seeded faults. Hence, despite the obvious importance of using real faults for program debugging experiments, most of the studies used seeded faults. Interestingly, 80% of the studies that use Java programs utilize real faults for their experiments. This means that most of the existing datasets that have real industrial faults suitable for multiple faults experiments are in Java programming language.

Due to the empirical and active nature of the MFL research domain, it is imperative to have a good evaluation metrics to measure a technique or approach effectiveness with respect to fault localization effectiveness. Out of the 20 identified evaluation metrics from the selected studies, we found out that the top three most used evaluation metrics are Expense score, Exam score, and Wasted effort with 15, 15, and 8 studies, respectively.

4.2. Challenges and direction for future work

The findings of this systematic review have profound implications for researchers who are working on MFL, since this study will allow them to uncover existing debugging approaches in the literature used in localizing multiple faults. Also, to aid in exploring the datasets, datasets programming languages, fault types, and also evaluation metrics used by the selected studies in localizing multiple faults. In this section, the identified research challenges and future research directions are outlined to guide new and veteran researchers in the field of research.

- **Challenges with OBA debugging approach:** We observed that the OBA debugging approach has gained a lot of attention in the last 10 years with 72.73% of the selected studies utilizing the method. However, various studies have hinted the shortcomings of using the method for localizing multiple faults, because more time needs to be spent in the localization process in comparison to parallel and MBA debugging approaches, and as outlined by other researchers, additional faults might be introduced in the software program during regression testing when OBA is utilized [2, 9, 76]. Therefore, its increased usage is alarming, with methods design to solve the problem having less attention such as parallel debugging approach and simultaneous debugging approach. Hence, in order to improve localization effectiveness in multiple fault contexts, perhaps, more studies in these two approaches (parallel and MBA debugging approaches) are of eminent importance.

- **Faults utilization:** Artificial faults (seeded faults), often used to replicate real faults behavior are the most utilized in MFL studies with 70.91% of the selected studies utilizing it. From the literature, real faults are consistently been adopted in the last few years for program debugging experiments (especially in the context of multiple faults), which legitimizes the experimental results by reducing bias and enhancing results generalization. For future works, we recommend the strict utilization of datasets that have real faults to meet up with the software industry standard and further encourage the use of fault localization techniques in the software industry.
- **Fault interference:** Fault interference is no doubt an inevitable factor as it occurs when more than one fault exists in a software program [2, 9, 27]. This phenomenon reduces fault localization techniques inferencing due to fault-to-failure complexity [29]. With the utilization of method such as parallel debugging, this phenomenon has been subsided with the aid of clustering algorithms for fault isolation. However, various studies indicated the lack of accuracy of these algorithms in isolating faults [11, 63]. Perhaps, better clustering algorithms are needed to resolve this issue which will help reduce the impact of fault interference and improve localization effectiveness.
- **Impact of clustering algorithms for fault isolation in parallel debugging:** On fault isolation, looking at the importance of clustering algorithms in the isolation of multiple faults and the lack of accuracy affecting the existing algorithms utilized in the literature, exploring machine learning algorithms might improve fault isolation accuracy and enhance both effectiveness and efficiency in the fault localization process in the research domain. Clustering algorithms are generally used when *fault-focused* clusters are to be generated, so as to aid in accurately measuring the fault-to-failure complexity between two failed test executions. In most of the existing studies, the key challenges are how to select the initial set of *fault-focused* clusters, the clustering algorithms to be used, and most importantly, the distance metrics to be used. However, most of the existing studies group failed test cases based on their execution profile similarity to generate the initial set of clusters. This approach is problematic because faults can be triggered in different ways. Some studies also estimate the number of clusters based on the number of failed test cases. However, this is also questionable because there is no clear correlation between the number of failed test cases and the number of faults in a program. Because each *fault-focused* cluster is supposed to target a single fault. Moreover, we observed that from the literature, distance metrics such as Euclidian distance, Jaccard distance, and Hamming distance have been used in the software fault localization research domain. These metrics where claimed to be inappropriate to use by Gao et al. [71] in measuring the *due-to* relationship between failed test cases. However, a study by Gao et al. [71] proposed a more advanced clustering method, with enhanced cluster estimation procedure and distance metrics. Hence, despite this, new and improved mechanisms are needed that will help in a more efficient and effective cluster estimation, in measuring the *due-to* relationships and distance between failed test executions.
- **Diversity in the subject programs used and evaluation:** Experiments on multilingual software programs are virtually nonexistence with multiple faults with respect to our selected studies. Hence, in order to address the issues of multiple faults, multilingual software programs are vital due to the current complexity and diversity of software used in the software industry [17]. We further recommend the use of statistical analysis to validate one's work. Looking at Table 9, only five studies (S10, S20, S26, S50, and S51) evaluate their work using statistical analysis method such as WSR. Hence, the utilization of such in evaluating one's work can be an added advantage.
- **Challenges with** MBA **debugging appraoch:** One of the key issue with MBA debugging approach is that the fault localization process can span to multiple debugging iterations. Ideally, a developer should be able to find all the faults in a single debugging iteration. However, when the fault search cap set by the developer is too low and the developer was not able to find all the faults in the first debugging iteration, a developer has to find and fixed the remaining faults in the subsequent iterations. This problem of multiple iterations in MBA debugging approach aid in creating a scenario like OBA debugging approach, whereby localizing and fixing of faults takes multiple iterations to be achieved. Based on the existing studies, this problem is not pervasive in MBA debugging approaches, but it is a concern that needs to be tackle through new solutions so as to aid developers in localizing all the faults in a single debugging iteration simultaneously.

Furthermore, other challenges in the literature such as fault introduced by missing code [10], fault localization on concurrency faults program [84], ties within fault localization ranking [85], and coincidental correctness [86] needs to be tackled and addressed.

4.3. *Threat to validity*

In order to have a comprehensive analysis of the results acquired from this review, the limitations of this review must be considered. The major threats to this SLR study validity are the inaccuracy of data extraction, biases on study selection, imbalance of study distribution, and biases on data synthesis. All these threats are discussed in this section.

4.3.1. *Incompleteness of study search*

In the quest of retrieving the studies, there may be relevant studies that might not be retrieved, which may affect the overall completeness of the study search. To mitigate this threat and to ensure that all relevant and potential studies have been covered, we conducted a broad search on five electronic databases where a large number of Journals, Conference, Workshop, and Symposium proceedings in the software engineering field are indexed. Furthermore, a manual collection of potential articles from a database such as Scopus was also included in the initial phase. This will ensure that relevant studies are not excluded due to the limitations of automatic search that can arise from different and vague search terms.

Moreover, the final studies selected were backward and forward reference searched to ensure that relevant studies are included. Although we took measures to improve the completeness of the study search, still the study may suffer from selection bias. This is because other libraries such as Citeceerx and EI Compendex were not considered.

4.3.2. *Bias on study selection*

To reduce bias by researchers with regards to the study selection process, a clear and well-defined inclusion, and exclusion criteria were formulated. Different researchers may have a distinct understanding of the IC/EC criteria, therefore, the study selection results of each researcher are likely to vary. In order to mitigate this bias, a pilot selection was performed to ensure that a consensus between the researchers is reached on the understanding of the study selection criteria. The potential mishandling of duplications is another threat as well. This threat may have slightly altered our results. Three cases of possible duplication were identified and were rigorously examined to uncover whether they are the same study.

Moreover, the final decision to select a study is done by the two researchers who conducted the search process. Hence, any disagreement that emerges between the two researchers will be resolved between them through discussion until a concrete agreement is found. The remaining researchers will review the final selected studies. In this SLR study, only peer-reviewed studies were included. However, there is a possibility that we might miss some important non-peer-reviewed study on MFL.

4.3.3. *Imbalance of study distribution over publication sources*

From Table 7 (distribution of studies with respect to their publication sources), 14.54% (8 out of 55) of the selected studies come from Workshop and Symposium proceedings. These studies may carry the biases of the Workshop/Symposium organizers and committee members. We did not tackle this kind of biases in this study, because there is no effective process to mitigate such biases. However, looking at Table 7, we found out that 50.91% (28 out of 55) of the selected studies come from Journals where most of which are highly ranked Journals such as Journal of Systems and Software (with 9 studies), Information and Software Technology (with 2 studies), Transactions on Software Engineering (with 2 studies) and so on. Hence, with a large number of the selected studies published in these Journals, the biases are mitigated to some extent.

4.3.4. *Inaccuracy of data extraction*

With regards to data extraction, bias may occur in this process which may affect the results classification and analysis of the selected studies. To mitigate this bias, the data items extracted in this systematic study were discussed among researchers and agreement on the meaning of each item was reached. Moreover, a pilot data extraction and selection were conducted among the researchers and disagreements on the results of the data were discussed and consensus was reached. Hence, the data items extracted were checked by two researchers where disagreements were discussed and resolved. With these measures taken to reduce bias, the accuracy of the extracted data items is further improved.

4.3.5. *Bias on data synthesis*

From the selected studies, not all the studies clearly describe the details of information that is to be extracted as data items. As a result, some information about data items had to be infer during the data synthesis. For example, a study may use a debugging approach to localize multiple faults without stating so or without giving much details to the readers on how they use it. Therefore, the researchers make the final conclusion on which debugging approach used by a study based on the nature of their experimental setup. Hence, potential ambiguities and bias can be mitigated or reduced.

5. Conclusion

The importance of software in our daily life cannot be underestimated. Persistently, most of our daily life activities have been partially automated (if not completely automated). Therefore, our dependency on software systems is increasing rapidly. However, with all the advancement of software in recent years, faults are almost inevitable. Locating these faults is difficult and costly which is why we have seen various fault localization techniques been proposed in the past two decades. Most of the previous studies have made a single fault assumption when a failure occurs. This assumption is problematic, as many faults can be the underlining courses of these failure(s). Of recent, researchers have competed in finding effective ways to localize multiple faults effectively which results in the introduction and utilization of various debugging approaches, techniques, clustering algorithms, and so on.

In this paper, we systematically reviewed the available literature on multiple fault localization (MFL). MFL is becoming one of the most critical issues in the software fault localization (SFL) research domain. Hence, significant research work is necessary for this domain to fill the existing gap between research and industry. An evidence-based methodology was adopted where 2617 articles were retrieved from our initial search. Based on our inclusion, exclusion, and quality criteria, 55 articles were finally selected for the study.

The result showed that an increasing amount of attention has been given to MFL since 2014, with 50.91% of the selected studies appeared in Journals, followed by Conferences, Symposium, and Workshops with 36.36%, 10.91%, and 1.82%, respectively. Also, the results from our analysis and synthesis revealed that there are three prominent multiple faults debugging approaches which are One-bug-at-a-time (OBA) debugging approach, parallel debugging approach, and multiple-fault-at-a-time (MBA) debugging approach. We observed that the OBA debugging approach followed by parallel debugging approach was largely been used for MFL research, with 72.73% and 20% of the selected studies utilizing them, respectively. Moreover, with respect to our quality assessments in this study, there has been a steady increase in the quality of papers on MFL from 2013. We further observed that programs from the Siemens test suite were still the most used. However, the utilization of Defect4j datasets in the MFL domain is on the rise in the last two years. With respect to datasets programming languages, datasets written in C were the most utilized followed by Java with 60% and 18.18% of the studies utilizing them, respectively. On fault types, we found out that despite the obvious advantages of using real faults and the current insistence by some veteran researchers to use it for program debugging experiments, most studies used seeded faults (artificial faults). Hence, with regards to evaluation metrics, the top three most utilized metrics were Expense score (15), Exam score (15), and Wasted effort (8), respectively.

In general, with the huge interest in MFL research from the research community and the recent consistency in publication in the domain, we expect more concrete solutions in the issue of multiple faults in years to come. Furthermore, some of the research challenges identified and some potential directions for future work were highlighted in detail in Section 4.2. Therefore, researchers' efforts must concentrate on these highlighted areas to effectively tackle the underlying challenges and propose workable solutions.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix

Table A1 highlights the authors' names, institutions, and country of institutions with respect to the individual selected studies.

Table A1
Author affiliation details.

Identifier	Authors	Institutions	Country of institution
S1	Wei	Northwestern Polytechnical University	China
	Han	Northwestern Polytechnical University	China
S2	Högerle	Lehrgebiet Programmiersysteme Fernuniversitat in Hagen	Germany
	Steimann	Lehrgebiet Programmiersysteme Fernuniversitat in Hagen	Germany
	Frenkel	Lehrgebiet Programmiersysteme Fernuniversitat in Hagen	Germany
S3	Laghari	Universiteit Antwerpen	Belgium
	Murgia	Universiteit Antwerpen	Belgium
	Demeyer	Universiteit Antwerpen	Belgium
S4	DiGiuseppe	University of California, Irvine	USA
	Jones	University of California, Irvine	USA
S5	Abreu	University of Porto	Portugal
	Gonzalez-Sanchez	Delft University of Technology	Netherlands
	Van Gemund	Delft University of Technology	Netherlands
S6	Lee	Software R&D center, Samsung Electronics	South Korea
	Kim	Sungkunkwan University	South Korea
	Lee	Sungkunkwan University	South Korea
S7	DiGiuseppe	University of California, Irvine	USA
	Jones	University of California, Irvine	USA
S8	Xia	Zhejiang University	China
	Gong	University of California, Berkeley	USA
	B	Singapore Management University	Singapore
	David Lo	Singapore Management University	Singapore
	Jiang	Singapore Management University	Singapore
	Zhang	Singapore Management University	Singapore
S9	Zhang	Beihang University	China
	Zheng	Beihang University	China
	Cai	Beihang University	China
S10	Zheng	Tianjin University	China
	Wang	Tianjin University	China
	Fan	Tianjin University	China
	Chen	Nantong University	China
	Yang	Western Michigan University	USA
S11	Zhang	Chinese Academy of Sciences	China
	Yan	Chinese Academy of Sciences	China
	Zhang	University of California, Irvine	USA
	Zhang	University of Chinese Academy of Sciences	China
	Chan	City University of Hong Kong	Hong Kong
	Zheng	Beihang University	China
S12	Sun	Fudan University	China
	Peng	Fudan University	China
	Li	Southeast University	China
	Wen	Nantong University	China
S13	Parsa	Iran University of Science and Technology	Iran
	Vahidi-Asl	Iran University of Science and Technology	Iran
	Zareie	Iran University of Science and Technology	Iran
S14	Wang	Beihang University	China
	Liu	Sun Yat-sen University	China
S15	Lamraoui	The Graduate University of Advance Studies	Japan
	Nakajima	The Graduate University of Advance Studies	Japan
S16	Dandan	Shanghai Academy of Spaceflight Technology	China
	Xiaohong	Harbin Institute of Technology	China
	Tiantian	Harbin Institute of Technology	China
	Peijun	Harbin Institute of Technology	China
	Yu	Harbin Institute of Technology	China
S17	Zhang	University of Notre Dame	USA
	Santelices	University of Notre Dame	USA
S18	Wang	Sun Yat-sen University	China
	Liu	Sun Yat-sen University	China
S19	Yu	Beihang University	China
	Bai	Beihang University	China
	Cai	Beihang University	China
S20	Naish	The University of Melbourne	Australia
	Neelofar	The University of Melbourne	Australia
	Ramamohanarao	The University of Melbourne	Australia
S21	Liu	University of Luxembourg	Luxembourg
	Lucia	University of Luxembourg	Luxembourg
	Nejati	University of Luxembourg	Luxembourg
	Briand	University of Luxembourg	Luxembourg
	Bruckmann	Delphi Automotive Systems	Luxembourg

(*continued on next page*)

Table A1 (*continued*)

Identifier	Authors	Institutions	Country of institution
S22	Xia	Zhejiang University	China
	Bao	Zhejiang University	China
	Lo	Singapore Management University	Singapore
	Li	Hengtian Software Ltd.	China
S23	Fu	East China University of Science and Technology	China
	Yu	East China University of Science and Technology	China
	Fan	East China University of Science and Technology	China
	Ji	East China University of Science and Technology	China
S24	Sun	Case Western Reserve University	USA
	Podgurski	Case Western Reserve University	USA
S25	Xiaobo	Beihang University	China
	Bin	Beihang University	China
	Jianxing	Beihang University	China
S26	Perez	University of Porto	Portugal
	Abreu	Palo Alto Research Center	USA
	Amorim	Federal University of Pernambuco	Brazil
S27	Aribi	University of Oran	Algeria
	Maamar	University of Artois	France
	Lazaar	University of Montpellier	France
	Lebbah	University of Oran	Algeria
	Loudni	University o Caen Normandy	France
S28	DiGiuseppe	University of California, Irvine	USA
	Jones	University of California, Irvine	USA
S29	Debroy	The University of Texas at Dallas	USA
	Wong	The University of Texas at Dallas	USA
S30	Abreu	Delft University of Technology	Netherlands
	Zoeteweij	Delft University of Technology	Netherlands
	Van Gemund	Delft University of Technology	Netherlands
S31	Xue	Texas Tech University	USA
	Namin	Texas Tech University	USA
S32	Huang	Nanjing University	China
	Wu	Southeast University	China
	Feng	Nanjing University	China
	Chen	Nanjing University	China
	Zhao	Nanjing University	China
S33	Steimann	Lehrgebiet Programmiersysteme Fernuniversitat in Hagen	Germany
	Frenkel	Lehrgebiet Programmiersysteme Fernuniversitat in Hagen	Germany
S34	Jeffrey	University of California, Riverside	USA
	Gupta	University of California, Riverside	USA
	Gupta	University of California, Riverside	USA
S35	Abreu	Delft University of Technology	Netherlands
	Zoeteweij	Delft University of Technology	Netherlands
	Van Gemund	Delft University of Technology	Netherlands
S36	Zhang	University of Texas at Austin	USA
	Li	University of Texas at Dallas	USA
	Zhang	University of Texas at Dallas	USA
	Khurshid	University of Texas at Austin	USA
S37	Xu	Zhejiang University	China
	Zhang	Chinese Academy of Sciences	China
	Chan	City University of Hong Kong	Hong Kong
	Tse	The University of Hong Kong	Hong Kong
	Li	Zhejiang University	China
S38	Perez	University of Porto	Portugal
	Abreu	University of Porto	Portugal
	Riboira	University of Porto	Portugal
S39	Wang	Nanjing University	China
	Gao	University of Texas at Dallas	USA
	Chen	Nanjing University	China
	Wong	University of Texas at Dallas	USA
	Luo	Nanjing University	China
S40	Abreu	University of Porto	Portugal
	Zoeteweij	IntelliMagic B.V	Netherlands
	Van Gemund	Delft University of Technology	Netherlands
S41	Naish	The University of Melbourne	Australia
	Neelofar	The University of Melbourne	Australia
	Ramamohanarao	The University of Melbourne	Australia
S42	Li	Georgia Institute of Technology	USA
	D'Amorim	Federal University of Pernambuco	Brazil
	Orso	Georgia Institute of Technology	USA
S43	Dean	Clemson University	USA
	Pressly	Clemson University	USA
	Malloy	Clemson University	USA
	Whitley	Clemson University	USA

(*continued on next page*)

Table A1 (*continued*)

Identifier	Authors	Institutions	Country of institution
S44	Liu	Beihang University	China
	Li	Beihang University	China
	Luo	Beihang University	China
S45	Neelofar	The University of Melbourne	Australia
	Naish	The University of Melbourne	Australia
	Lee	The University of Melbourne	Australia
	Ramamohanarao	The University of Melbourne	Australia
S46	Lucia	Singapore Management University	Singapore
	Lo	Singapore Management University	Singapore
	Jiang	Singapore Management University	Singapore
	Thung	Singapore Management University	Singapore
	Budi	Singapore Management University	Singapore
S47	Ghandehari	University of Texas at Arlington	USA
	Lei	University of Texas at Arlington	USA
	Kacker	National Institute of Standards and Technology	USA
	Kuhn	University of Texas at Arlington	USA
	Xie	North Carolina State University	USA
	Kung	University of Texas at Arlington	USA
S48	Wang	Anhui Polytechnic University	China
	Huang	Nanjing University of Aeronautics and Astronautics	China
	Fang	Nanjing University of Aeronautics and Astronautics	China
	Li	Nanjing University of Aeronautics and Astronautics	China
S49	Zakari	Kano University of Science and Technology, Wudil	Nigeria
	Lee	University of Malaya	Malaysia
	Chong	Monash University	Malaysia
S50	Gao	University of Texas at Dallas	USA
	Wong	University of Texas at Dallas	USA
S51	Wong	University of Texas at Dallas	USA
	Debroy	University of Texas at Dallas	USA
	Gao	University of Texas at Dallas	USA
	Li	University of Texas at Dallas	USA
S52	Wong	University of Texas at Dallas	USA
	Debroy	University of Texas at Dallas	USA
	Xu	Dakota State University	USA
S53	Wong	University of Texas at Dallas	USA
	Debroy	University of Texas at Dallas	USA
	Golden	University of Texas at Dallas	USA
	Xu	University of Texas at Dallas	USA
	Thuraisingham	University of Texas at Dallas	USA
S54	Parsa	Iran University of Science and Technology	Iran
	Vahidi-Asl	Iran University of Science and Technology	Iran
	Maryam	Iran University of Science and Technology	Iran
S55	Gao	University of Texas at Dallas	USA
	Wong	University of Texas at Dallas	USA
	Chen	Nanjing University	China
	Wang	Nanjing University	China

References

[1] T. Shu, et al., Fault localization based on statement frequency, Inf. Sci. 360 (2016) 43–56.

[2] N. DiGiuseppe, J.A. Jones, Fault density, fault types, and spectra-based fault localization, Empir. Softw. Eng. 20 (4) (2015) 928–967.

[3] A. Zakari, S.P. Lee, I.A.T. Hashem, A community-based fault isolation approach for effective simultaneous localization of faults, IEEE Access 7 (2019) 50012–50030.

[4] A. Zakari, S.P. Lee, Simultaneous isolation of software faults for effective fault localization, 2019 IEEE 15th International Colloquium on Signal Processing & Its Applications (CSPA), IEEE, 2019.

[5] J.A. Jones, M.J. Harrold, J. Stasko, Visualization of test information to assist fault localization, in: Proceedings - International Conference on Software Engineering, 2002.

[6] W.E. Wong, Y. Qi, BP neural network-based effective fault localization, Int. J. Softw. Eng. Knowl. Eng. 19 (4) (2009) 573–597.

[7] W.E. Wong, et al., Effective software fault localization using an rbf neural network, IEEE Trans. Reliab. 61 (1) (2012) 149–169.

[8] W. Zheng, D.S. Hu, J. Wang, Fault localization analysis based on deep neural network, Math. Problems Eng. 2016 (2016).

[9] X. Xue, A.S. Namin, How significant is the effect of fault interactions on coverage-based fault localizations? 2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, IEEE, 2013.

[10] W.E. Wong, et al., A survey on software fault localization, IEEE Trans. Soft. Eng. 42 (8) (2016) 707–740.

[11] W.E. Wong, V. Debroy, A Survey of Software Fault Localization, 9, Department of Computer Science, University of Texas at Dallas, 2009 Tech. Rep. UTDCS-45.

[12] M.A. Alipour, Automated Fault Localization Techniques: A Survey, State University, Oregon, 2012.

[13] D. Pal, R. Mohiuddin, Automated Bug Localization of Software Programs: Aa Survey Report1, 2013.

[14] A. Perez, R. Abreu, E. Wong, *A Survey on Fault Localization* Techniques, *A Survey on Fault Localization* Techniques, 88, 2014 Cited on pages iv and.

[15] P. Parmar, M. Patel, *Software fault localization: a survey*, Int. J. Comput. Appl. 154 (9) (2016).

[16] Souza, H.A., M.L. Chaim, and F. Kon, Spectrum-based software fault localization: a survey of techniques, advances, and challenges. arXiv preprint , 2016.

[17] A. Zakari, et al., Software Fault Localization: A Systematic Mapping Study, IET Software, 2018.

[18] S. Keele, Guidelines for Performing Systematic Literature Reviews in Software Engineering, EBSE, 2007 *Technical report, Ver. 2.3 EBSE Technical Report*.

[19] Y. Xiao, M. Watson, Guidance on conducting a systematic literature review, J. Plann. Educ. Res. (2017) 0739456X17723971.

[20] B. Kitchenham, et al., Systematic literature reviews in software engineering–a tertiary study, Inf. Softw. Technol. 52 (8) (2010) 792–805.

[21] S. Ouhbi, et al., Requirements engineering education: a systematic mapping study, Req. Eng. 20 (2) (2015) 119–138.

[22] K.A. Alam, et al., Impact analysis and change propagation in service-oriented enterprises: a systematic review, Inf. Syst. 54 (2015) 43–73.

[23] , Computer Science Conference Rankings, CORE, 2015.

[24] Z. Wei, B. Han, Multiple-Bug oriented fault localization: a parameter-based combination approach, Software Security and Reliability-Companion (SERE-C), 2013 IEEE 7th International Conference on, IEEE, 2013.

[25] W. Högerle, F. Steimann, M. Frenkel, More debugging in parallel, 2014 IEEE 25th International Symposium on Software Reliability Engineering, IEEE, 2014.

[26] G. Laghari, A. Murgia, S. Demeyer, Fine-tuning spectrum based fault localisation with frequent method item sets, in: Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, ACM, 2016.

[27] N. DiGiuseppe, J.A. Jones, On the influence of multiple faults on coverage-based fault localization, in: Proceedings of the 2011 International Symposium on Software Testing and Analysis, ACM, 2011.
[28] R. Abreu, A. Gonzalez-Sanchez, A.J. van Gemund, Exploiting count spectra for Bayesian fault localization, in: Proceedings of the 6th International Conference on Predictive Models in Software Engineering, ACM, 2010.
[29] J. Lee, J. Kim, E. Lee, Enhanced fault localization by weighting test cases with multiple faults, in: Proceedings of the International Conference on Software Engineering Research and Practice (SERP), The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2016.
[30] X. Xia, et al., Diversity maximization speedup for localizing faults in single-fault and multi-fault programs, Autom. Softw. Eng. 23 (1) (2016) 43–75.
[31] X.-.Y. Zhang, Z. Zheng, K.-.Y. Cai, Exploring the usefulness of unlabelled test cases in software fault localization, J. Syst. Softw. 136 (2018) 278–290.
[32] Y. Zheng, et al., Localizing multiple software faults based on evolution algorithm, J. Syst. Softw. 139 (2018) 107–123.
[33] L. Zhang, et al., A theoretical analysis on cloning the failed test cases to improve spectrum-based fault localization, J. Syst. Softw. 129 (2017) 35–57.
[34] X. Sun, et al., IPSETFUL: an iterative process of selecting test cases for effective fault localization by exploring concept lattice of program spectra, Front. Comput. Sci. 10 (5) (2016) 812–831.
[35] S. Parsa, M. Vahidi-Asl, F. Zareie, Statistical based slicing method for prioritizing program fault relevant statements, Comput. Inform. 34 (4) (2016) 823–857.
[36] X. Wang, Y. Liu, Fault localization using disparities of dynamic invariants, J. Syst. Softw. 122 (2016) 144–154.
[37] S.-.M. Lamraoui, S. Nakajima, A formula-based approach for automatic fault localization of multi-fault programs, J. Inf. Process. 24 (1) (2016) 88–98.
[38] D. Gong, et al., State dependency probabilistic model for fault localization, Inf. Softw. Technol. 57 (2015) 430–445.
[39] Y. Zhang, R. Santelices, Prioritized static slicing and its application to fault localization, J. Syst. Softw. 114 (2016) 38–53.
[40] X. Wang, Y. Liu, Automated fault localization via hierarchical multiple predicate switching, J. Syst. Softw. 104 (2015) 69–81.
[41] Z.X. Yu, et al., Does the failing test execute a single or multiple faults? An approach to classifying failing tests, in: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, vol 1, 2015, pp. 924–935.
[42] L. Naish, K. Ramamohanarao, Multiple bug spectral fault localization using genetic programming, Software Engineering Conference (ASWEC), 2015 24th Australasian, IEEE, 2015.
[43] B. Liu, et al., Localizing multiple faults in simulink models, Software Analysis, Evolution, and Reengineering (SANER), 2016 IEEE 23rd International Conference on, IEEE, 2016.
[44] X. Xia, et al., " Automated Debugging Considered Harmful" Considered Harmful: A User Study Revisiting the Usefulness of Spectra-Based Fault Localization Techniques with Professionals Using Real Bugs from Large Systems, 2016.
[45] W. Fu, et al., Test case prioritization approach to improving the effectiveness of fault localization, Software Analysis, Testing and Evolution (SATE), International Conference on, IEEE, 2016.
[46] S.-.F. Sun, A. Podgurski, Properties of effective metrics for coverage-based statistical fault localization, Software Testing, Verification and Validation (ICST), 2016 IEEE International Conference on, IEEE, 2016.
[47] Y. Xiaobo, L. Bin, L. Jianxing, The failure behaviors of multi-faults programs: an empirical study, Software Quality, Reliability and Security Companion (QRS-C), 2017 IEEE International Conference on, IEEE, 2017.
[48] A. Perez, R. Abreu, M. D'Amorim, Prevalence of single-fault fixes and its impact on fault localization, Software Testing, Verification and Validation (ICST), 2017 IEEE International Conference on, IEEE, 2017.
[49] N. Aribi, et al., Multiple fault localization using constraint programming and pattern mining, Tools with Artificial Intelligence (ICTAI), 2017 IEEE 29th International Conference on, IEEE, 2017.
[50] N. DiGiuseppe, J.A. Jones, Fault interaction and its repercussions, Software Maintenance (ICSM), 2011 27th IEEE International Conference on, IEEE, 2011.
[51] V. Debroy, W.E. Wong, Insights on fault interference for programs with multiple bugs, Software Reliability Engineering, 2009. ISSRE'09. 20th International Symposium on, IEEE, 2009.
[52] R. Abreu, P. Zoeteweij, A.J. Van Gemund, Spectrum-based multiple fault localization, Automated Software Engineering, 2009. ASE'09. 24th IEEE/ACM International Conference on, IEEE, 2009.
[53] Y. Huang, et al., An empirical study on clustering for isolating bugs in fault localization, Software Reliability Engineering Workshops (ISSREW), 2013 IEEE International Symposium on, IEEE, 2013.
[54] F. Steimann, M. Frenkel, Improving coverage-based localization of multiple faults using algorithms from integer linear programming, 2012 IEEE 23rd International Symposium on Software Reliability Engineering, IEEE, 2012.
[55] D. Jeffrey, N. Gupta, R. Gupta, Effective and efficient localization of multiple faults using value replacement, Software Maintenance, 2009. ICSM 2009. IEEE International Conference on, IEEE, 2009.
[56] R. Abreu, P. Zoeteweij, A.J. van Gemund, Localizing software faults simultaneously, 2009 Ninth International Conference on Quality Software, IEEE, 2009.
[57] M. Zhang, et al., Boosting spectrum-based fault localization using pagerank, in: Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis, ACM, 2017.
[58] J. Xu, et al., A general noise-reduction framework for fault localization of Java programs, Inf. Softw. Technol. 55 (5) (2013) 880–896.
[59] A. Perez, R. Abreu, A. Riboira, A dynamic code coverage approach to maximize fault localization efficiency, J. Syst. Softw. 90 (2014) 18–28.
[60] Y. Wang, et al., WAS: a weighted attribute-based strategy for cluster test selection, J. Syst. Softw. 98 (2014) 44–58.
[61] R. Abreu, P. Zoeteweij, A.J. Van Gemund, Simultaneous debugging of software faults, J. Syst. Softw. 84 (4) (2011) 573–586.
[62] N. Neelofar, L. Naish, K. Ramamohanarao, Spectral-based fault localization using hyperbolic function, Software 48 (3) (2018) 641–664.
[63] X. Li, M. d'Amorim, A. Orso, Iterative user-driven fault localization, Haifa Verification Conference, Springer, 2016.
[64] B.C. Dean, et al., A linear programming approach for automated localization of multiple faults, in: Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering, IEEE Computer Society, 2009.
[65] A. Liu, L. Li, J. Luo, Automated program debugging for multiple bugs based on semantic analysis, International Workshop on Structured Object-Oriented Formal Language and Method, Springer, 2015.
[66] N. Neelofar, et al., Improving spectral-based fault localization using static analysis, Software (2017).
[67] L. Lucia, et al., Extended comprehensive study of association measures for fault localization, J. Softw. 26 (2) (2014) 172–219.
[68] L.S. Ghandehari, et al., A combinatorial testing-based approach to fault localization, IEEE Trans. Soft. Eng. (2018).
[69] Y. Wang, et al., Spectrum-Based fault localization via enlarging non-fault region to improve fault absolute ranking, IEEE Access 6 (2018) 8925–8933.
[70] A. Zakari, S.P. Lee, C.Y. Chong, Simultaneous localization of software faults based on complex network theory, IEEE Access (2018).
[71] R. Gao, W.E. Wong, MSeer-an advanced technique for locating multiple bugs in parallel, IEEE Trans. Soft. Eng. (2017).
[72] W.E. Wong, et al., The DStar method for effective software fault localization, IEEE Trans. Reliab. 63 (1) (2014) 290–308.
[73] W.E. Wong, V. Debroy, D. Xu, Towards better fault localization: a crosstab-based statistical approach, IEEE Trans. Syst. Man Cybern. Part C 42 (3) (2012) 378–396.
[74] S. Parsa, M. Vahidi-Asl, M. Asadi-Aghbolaghi, Hierarchy-Debug: a scalable statistical technique for fault localization, Softw. Qual. J. 22 (3) (2014) 427–466.
[75] R. Gao, et al., Effective software fault localization using predicted execution results, Softw. Qual. J. 25 (1) (2017) 131–169.
[76] J.A. Jones, M.J. Harrold, Empirical evaluation of the tarantula automatic fault-localization technique, 20th IEEE/ACM International Conference on Automated Software Engineering, ASE 2005, 2005.
[77] J.A. Jones, J.F. Bowring, M.J. Harrold, Debugging in parallel, in: Proceedings of the 2007 International Symposium on Software Testing and Analysis, ACM, 2007.
[78] A. Zakari, S.P. Lee, Parallel debugging: an investigative study, J. Softw. (2019) e2178.
[79] C. Gong, et al., Factorising the multiple fault localization problem: adapting single–fault localizer to multi-fault programs, 2012 19th Asia-Pacific Software Engineering Conference, IEEE, 2012.
[80] G. Gousios, A. Zaidman, A dataset for pull-based development research, in: Proceedings of the 11th Working Conference on Mining Software Repositories, ACM, 2014.
[81] J.H. Andrews, L.C. Briand, Y. Labiche, Is mutation an appropriate tool for testing experiments? in: Proceedings of the 27th International Conference on Software Engineering, ACM, 2005.
[82] J.H. Andrews, et al., Using mutation analysis for assessing and comparing testing coverage criteria, IEEE Trans. Soft. Eng. (8) (2006) 608–624.
[83] R. Just, et al., Are mutants a valid substitute for real faults in software testing? in: Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, ACM, 2014.
[84] A. Farzan, P. Madhusudan, F. Sorrentino, Meta-analysis for atomicity violations under nested locking, Computer Aided Verification, Springer, 2009.
[85] X. Xu, et al., Ties within fault localization rankings: exposing and addressing the problem, Int. J. Softw. Eng. Knowl. Eng. 21 (06) (2011) 803–827.
[86] A. Bandyopadhyay, Mitigating the effect of coincidental correctness in spectrum based fault localization, Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on, IEEE, 2012.