Search-based Fault Localisation: A Systematic Mapping Study

Plinio S. Leitao-Junior[a,b,*], Diogo M. Freitas[a], Silvia R. Vergilio[c], Celso G. Camilo-Junior[a], Rachel Harrison[b]

[a] *Instituto de Informática, Universidade Federal de Goiás,*
Alameda Palmeiras, Quadra D, Câmpus Samambaia, Goiânia, Goiás, Brazil
[b] *School of Computing, Engineering and Maths, Oxford Brookes University,*
Wheatley Campus, OX33 1HX, Wheatley, Oxford, United Kingdom
[c] *Departamento de Informática, Universidade Federal do Paraná,*
Rua Cel. Francisco Heráclito Santos 100, Jardim das Américas, Curitiba, Paraná, Brazil

Abstract

Context: Software *Fault Localisation* (FL) refers to finding faulty software elements related to failures produced as a result of test case execution. This is a laborious and time consuming task. To allow FL automation search-based algorithms have been successfully applied in the field of Search-Based Fault Localisation (SBFL). However, there is no study mapping the SBFL field to the best of our knowledge and we believe that such a map is important to promote new advances in this field.
Objective: To present the results of a mapping study on SBFL, by characterising the proposed methods, identifying sources of used information, adopted evaluation functions, applied algorithms and elements regarding reported experiments.
Method: Our mapping followed a defined process and a search protocol. The conducted analysis considers different dimensions and categories related to the main characteristics of SBFL methods.
Results: All methods are grounded on the coverage spectra category. Overall the methods search for solutions related to suspiciousness formulae to identify possible faulty code elements. Most studies use evolutionary algorithms, mainly Genetic Programming, by using a single-objective function. There is little investigation of real-and-multiple-fault scenarios, and the subjects are mostly written in C and Java. No consensus was observed on how to apply the evaluation metrics.
Conclusions: Search-based fault localisation has seen a rise in interest in the past few years and the number of studies has been growing. We identified some research opportunities such as exploring new sources of fault data, exploring multi-objective algorithms, analysing benchmarks according to some classes of

[*] Corresponding author

Email addresses: `plinio@inf.ufg.br` (Plinio S. Leitao-Junior), `diogom42@gmail.com` (Diogo M. Freitas), `silvia@inf.ufpr.br` (Silvia R. Vergilio), `celso@inf.ufg.br` (Celso G. Camilo-Junior), `rachel.harrison@brookes.ac.uk` (Rachel Harrison)

January 17, 2020

faults, as well as, the use of a unique definition for evaluation measures.

Keywords: Meta-heuristic algorithms; Search-based fault localisation; Systematic mapping.

1. Introduction

In recent decades, our reliance on software in all areas of human activity has increased significantly as a result of the increasing use of computer-based systems. This causes growing demands for quality and productivity, from the point of view of both the production processes and the generated product.

It is generally accepted that it is not possible to create perfect software, and mistakes or introduced defects (faults) occur and may be largely unavoidable [1].Faults are constantly introduced and fixed during the software production and maintenance cycles. The presence of faults in software may stem from a variety of factors including, but not limited to, changes in user's needs, misunderstanding of software requirements, inadequate software design, low-quality code, poor documentation, and mistakes in the coding phase. Software can still contain faults, even after completion of extensive testing, and failures experienced after software delivery are addressed by corrective maintenance [2]. Therefore, one of the main goals during software development and evolution is to remove as many faults in the software as possible without introducing new ones while doing so.

According to the Software Engineering Guide Body of Knowledge (SWEBOK) [2], software maintenance provides unique technical and management challenges for software engineers, such as trying to find one fault in a software system that contains a large number of lines of code and was developed by another software engineer. In this sense *software debugging* is a process aimed at finding and resolving faults that prevent the correct operation of computer software or of systems thereof.

Due to the increasing size and complexity of software projects, finding faults has become a more onerous and time-consuming task [3]. Software *Fault Localisation* (FL) is a vital process which refers to finding the faulty software elements (*e.g.* statement, line or block of code) related to failures that were revealed on the execution of software testing activities. Such a process can be laborious and time consuming when it is done manually as the complexity of software projects increases. Therefore, one of the main challenges of FL activities is to decrease the human effort by reducing the amount of code analysed until the software faults can be precisely located. Research into FL deals mainly with the problem of developing techniques to automate (or semi-automate) the process of locating software faults. To this end, we can find in the literature [4] different methods to help software engineering practitioners who often spend a significant amount of time and effort on debugging [5]. Among such methods, search-based methods have received increasing attention and a field of research has emerged, named *Search-based Fault Localisation* (SBFL).

In the SBFL field the FL problem is treated as an optimisation problem and search-based algorithms are used to automate (or partially automate) FL solutions. SBFL researchers usually apply evolutionary algorithms such as Genetic Programming, to derive metrics in order to measure the odds of each program element being faulty. Each individual in the population represents a candidate suspiciousness formula to solve the problem and the population is a set of solutions which evolves to achieve better equations to calculate how suspicious each software element is. A classical example is to rank the software statements with respect to their fault-proneness by applying an approach based on a Genetic Algorithm, but to the best of our knowledge there is no effort to provide an overall analysis of the SBFL methods in the literature.

In order to propose new SBFL methods that reduce the fault localisation effort, and to investigate how they are employed and evaluated, we need to examine and characterise existing methods. Considering this fact and to contribute to the development of the SBFL field, this paper provides results of a mapping study on the SBFL methods. The overall objective is to provide a study of the research on SBFL methods to systematically identify, analyse, and describe the state-of-art advances in the field.

In our mapping we followed a research plan, according to guidelines of Kitchenham *et al.* [6], including research questions, inclusion and exclusion criteria, construction of the search string and selection of known search databases. We found 14 primary studies, which are analysed considering the following dimensions: i) main fora and frequency of publications over the years; ii) research interests addressed in the field; iii) main characteristics of the proposed methods such as used algorithms, search process aspects and evaluation functions used; and iv) evaluation aspects regarding baselines used in the evaluations, identified benchmarks and evaluation measures.

As a contribution of our mapping we also discuss the main gaps we identified by analysing the found studies. They constitute research opportunities to guide future research in the field.

The paper is organised as follows. Section 2 reviews FL background and related work. Section 3 describes the protocol and procedure adopted in our mapping. The search process and the data extraction are in Sections 4 and 5 respectively. The main results and findings are analysed in Subsections 5.1 to 5.5, which provides answers to our research questions. Section 6 summarises our finding, by presenting gaps and trends identified and derived research opportunities. Section 7 details the main threats to validity of our results and how they were mitigated. Section 8 concludes the paper.

2. Background

The terms *error*, *fault* (*defect*), and *failure* are defined, respectively, as "erroneous state of the system", "defect in a system or a representation of a system that if executed/activated could potentially result in an error", and "an externally visible deviation from the system's specification" [7]. The Standard IEEE 1044 (2009) [8] states that "a failure may be caused by (and thus indicate the presence of) a fault" and "a fault may cause one or more failures". We adhere to this terminology in this paper.

Spectra-based analysis refers to a group of FL methods that use a program spectrum to find the location of the fault in the given program that causes certain tests to fail. Repps *et al.* initially hypothesise a strong correlation between spectra differences and faults [9], such as a correlation between distinct spectra for a faulty program and the correct version on the same input and a high fault count [10].

A *program spectrum* is an execution profile that indicates which parts of a program are active during a run [11], that can be applied as a heuristic for understanding the magnitude of the behavioural changes between program versions [9]. The most widely used type of program spectrum is the combination of code coverage and the test results: which code elements were executed (or not executed) by test cases that have passed (or failed).

In general, automation initiatives for FL propose formulae to calculate the odds of faulty program elements, and a number of spectrum-based formulae have been proposed in different studies as well as comparisons among them (*e.g.* [12, 13, 14, 11]).

2.1. Search-based Fault Localisation

Search-based Software Engineering (SBSE) is the name given to a body of work in which search-based optimisation is applied to Software Engineering [15]. Harman and Jones argue that like other engineering disciplines, Software Engineering is typically concerned with near optimal solutions or solutions which fall within a specified acceptable tolerance and these are the very factors which make robust metaheuristic search-based optimisation techniques readily applicable [16]. As defined by the authors, it is possible to apply metaheuristic search to a large body of software engineering problems, where natural representations, objective (fitness) functions and operators suggest themselves. For instance, software testing is an essential part of software engineering, and therefore testing problems can be modelled as search-based problems (*e.g.* sample data are selected from the program input domains which are in general infinite).

Search-based Fault Localisation (SBFL) is a research field that applies the SBSE paradigm to the fault localisation problem. The optimisation algorithm exploits the search space such that each element of this space denotes a candidate solution related to a potential fault location. It means SBFL research field essentially can cope with a search process to more precisely locate software faults.

In this sense there are intrinsic questions related that are typical of optimisation; two examples of *SBFL problems* are: What is the best ranking of software elements with respect to faulty ones when failures are revealed? What are the best formulae to calculate the suspiciousness of faults with respect to elements of a particular program? Note that such questions look for their answers in distinct search spaces, which are, respectively: (1) the whole set of program elements, which may become larger as the software complexity increases; and

(2) all valid formulae composed of variables and mathematical operators selected to build suspiciousness measures. The scientific community has become increasingly interested in the SBFL research field in recent years, specifically on applying metaheuristic algorithms to guide the search process.

2.2. Systematic Mapping Study

Kitchenham *et al.* [6] define a *systematic mapping study* (SMS), or *scoping study*, as a study whose objective is to provide a wide overview of the research area, to establish if research evidence exists on a topic and to provide an indication of the quantity of the evidence. Brereton *et al.* [17] highlight that systematic mapping is useful to establish the context of a review as well as to assist in the definition of research questions and selection criteria.

In this context there has been renewed interest in proposals for fault localisation methods, and as a result literature-reviewing papers of such research area have been published [4, 18] and the number of papers has increased since 2001 [4]. The present systematic mapping study deals with the proposition and evaluation of fault localisation techniques that are based on metaheuristic search. The focus of the metaheuristic search impacts the decision to include or not primary studies as relevant papers to the mapping; if so it means that the optimisation algorithm exploits the search space such that each element of this space is a candidate solution that indicates a potential defect's location. In summary the relevant papers directly cope with the question: how effective are techniques to precisely locate software faults.

3. Planning of the systematic mapping study

Following the guidelines of Kitchenham *et al.* [6] we created a protocol and structured our mapping study process into seven stages as illustrated in Figure 1. Such stages are based on [19, 20] and are briefly introduced below:

1. *the need and relevance* motivates the mapping study and states the research questions (Subsection 3.1);

2. *planning of the study* refers to the main steps needed to carry out the mapping study and outlines its structure (this section);

3. *search for primary studies* seeks relevant studies by following a search strategy (Section 4);

4. *inclusion and exclusion of primary studies* defines inclusion and exclusion criteria and strategies aimed at analysing the found studies by flagging them as relevant (or non relevant) to the mapping process (Subsection 4.1);

5. *data extraction* refers to collecting data from the relevant studies by applying systematic strategies; *e.g.* a classification schema is defined for guiding the data extraction (Section 5).

6. *data classification and visualisation* categorises the relevant studies and organises the classified data in order to present them as a map using charts and diagrams (Subsections 5.1 to 5.5).

To reduce any bias, *dealing with validity issues* (in Figure 1) refers to the systematic way of reducing the threats to validity on each process stage, *i.e.* the actions that have been taken to increase the reliability of the process. For instance on *the need and relevance* we carried out a systematic search by looking for similar-focused literature review papers aiming at increasing confidence on the novelty of this study. Discussion about the threats are presented in Section 7.

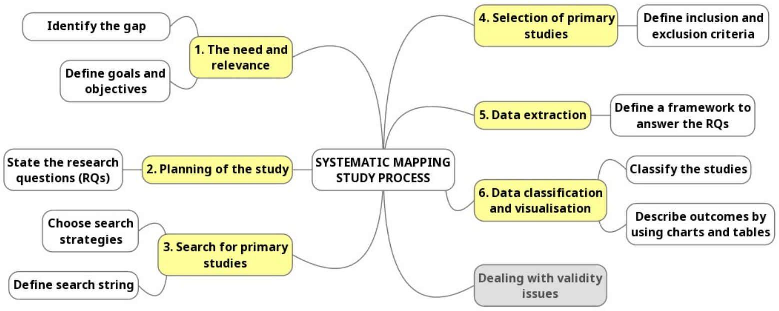

Identify the gap
Define goals and objectives
1. The need and relevance
State the research questions (RQs)
2. Planning of the study
SYSTEMATIC MAPPING STUDY PROCESS
Choose search strategies
3. Search for primary studies
Define search string
4. Selection of primary studies
Define inclusion and exclusion criteria
5. Data extraction
Define a framework to answer the RQs
6. Data classification and visualisation
Classify the studies
Describe outcomes by using charts and tables
Dealing with validity issues

Figure 1: Systematic mapping study process.

The following sections report the stages of the present mapping study as well as the research opportunities revealed by the study.

3.1. The need and relevance of the study

This section addresses the need and relevance of the present systematic mapping study which is focused on the SBFL research field. Firstly, we investigate whether other studies exist which pursue the same goal.

We performed a search looking for literature review studies from last ten years until June 2019. The choice of databases and the structure of the search string were based on Petersen *et al.*'s mapping study which investigates how systematic mapping processes have been executed in software engineering [20].

The search was carried out on the databases of IEEE Xplore, ACM, Scopus, as well as Inspec/Compendex (Engineering village) by applying the search string to the Metadata *title*, *abstract* and *keywords*. The search terms were grouped into three sets:

- The scope: *("software" OR "program" OR "programs") AND*
- The concept that is going to be observed: *("fault localisation" OR "fault localization" OR "defect localisation" OR "defect localization") AND*

- The process of classification and categorisation: *("systematic mapping" OR "systematic map" OR "systematic mapping study" OR "systematic mapping studies" OR "systematic review" OR "literature review" OR "survey")*

To identify the relevant studies, the search results were analysed by the two first authors of this paper. The analysis was based on titles and abstracts, as well as full-text reading. As a consequence, the reasons to flag studies as excluded were: conference proceedings, which also appear in Scopus and Inspec/Compendex results as publications (10 papers); secondary studies not related to fault localisation (7 papers); papers that are not secondary studies (10 papers); and studies that are not written in English (6 papers). The literature review studies that were flagged as relevant are listed in Table 1.

Table 1: Literature review studies of fault localisation.

#ID	Authors	Title	Source title	Year
S1	Zakari, Lee, Alam and Ahmad [18]	Software fault localisation: a systematic mapping study	IET Software	2019
S2	Wong, Gao, Li, Abreu and Wotawa [4]	A survey on software fault localization	IEEE Transactions on Software Engineering	2016
S3	Agarwal and Agrawal [21]	Fault-localization Techniques for Software Systems: A Literature Review	SIGSOFT Softw. Eng. Notes	2014

We use the terms *technique* and *method* with the same meaning, which denote a search-based solution to the fault localisation problem.

Regarding the analysis process of related secondary studies, we applied a set of comparison attributes over the studies aiming at comparing studies listed in Table 1 against the present mapping as follows.

A01: Are research questions presented?

A02: What is the period covered by the secondary study?

A03: Is the search string shown?

A04: Are the search databases listed?

A05: Are the inclusion and exclusion criteria defined?

A06: Are the inclusion and exclusion criteria justified?

A07: Are the selected papers explicitly identified?

A08: Are the papers that apply metaheuristic search identified?

A09: Are the recent papers that apply metaheuristic search included?

A10: Are research questions explicitly answered?

A11: Is there a crossover of answers from different research questions?

A12: Are research gaps and opportunities identified and presented?

Table 2: Comparison attributes over secondary studies.

#ID	A01	A02	A03	A04	A05	A06	A07	A08	A09	A10	A11	A12
S1	✓	2006-2017	✓	✓	✓	✗	✗	✗	✗	✓	✓	✓
S2	✗	1977-2014	✗	✗	✗	✗	✓	✗	✗	✗	✗	✓
S3	✗	2007-2013	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗
This mapping	✓	2001-2019	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

✓: Yes, ✗: No

We analyse each secondary study regarding comparison attributes as shown in Table 2. The present mapping fulfills all the attributes listed in the table.

Research questions (RQs) make the goals and contributions of secondary studies more objective and systematic. There are three attributes of the research questions, namely A01, A10, and A11: Studies S2 and S3 are not oriented by RQs and they do not meet any such attributes. Regarding the systematisation of the search process, Studies S2 and S3 also fail to present the search strings and search databases (Attributes A03 and A04, respectively).

On the selection of primary studies (Attributes A05 and A06), Studies S2 and S3 do not apply inclusion and exclusion criteria to select studies. Study S1 states inclusion and exclusion criteria but does not justify them, so that includes a validity threat to the selection process. Another important threat is the identification of selected studies (Attribute A07), as that makes the response process to the research questions reproducible and able to be evaluated: Study S1 does not meet that attribute.

Attribute A08 treats the identification of primary studies that apply metaheuristic search, i.e. the focus of the present mapping: search-based fault localisation (SBFL). Study S3 does not categorise the selected primary studies but only lists them by year. Studies S1 and S2 have similar ways to classify fault localisation (FL) methods such as spectrum-based, statistics-based, program state-based, machine learning-based techniques and hybrid. However, such studies do not address SBFL as a category of FL techniques. Study S2 does not select a primary study entitled ”Evolving fault location techniques based on human competitive spectra” and authored by Yoo et al. in 2012, which is an important contribution to SBFL's research field. In addition, Study S1 does not explicitly state what primary studies are the selected ones (only deals with the number of papers).

Regarding recent papers that apply metaheuristic search (Attribute A09), Studies S2 and S3 do not meet this attribute as their search period ends in 2014 and 2013 respectively. Study S1 covers up to 2017 but fails to meet Attribute A09 such year, as S1 selects only one paper on both categories machine learning-based techniques and hybrid (these categories could match some of the studies on SBFL). Finally only Studies S1 and S2 identify and present research gaps and opportunities (Attribute A12), which are expected findings from secondary studies.

Therefore, we did not identify any literature review papers that specifically focus on the SBFL research field nor categorisation scheme for SBFL-based methods. Such a finding reveals a gap for further efforts aiming to map the research area and to apply a classification schema to the published methods.

3.2. Research questions

Since the definition of research questions delimits the research scope and the purpose of systematic mappings is to establish an overview of the research field and to identify the number and types of research conducted so far, this study is guided by research questions (RQs) that pursue the aim:

RQ-1: *How has the number and the frequency of publications evolved over the years?* **Rationale:** this question aims to assess the relevance and activity of this topic in the SBSE community as well as its evolution in terms of the number and constancy of publications.

RQ-2: *What venues has the research on SBFL methods been published in?* **Rationale:** this question helps to identify the most preferred fora aiming to figure out what venues value the research field and to provide researchers with information concerning the best places to publish their research.

RQ-3: *What investigations and data are addressed by the studies?* **Rationale:** this question is aimed at identifying the research interests of the studies, the sources of fault data and the main targets of their research questions. To cope with this question, three sub-questions were formed:

RQ-3.1: *What data are considered as sources of faults to be located?*

RQ-3.2: *What do primary studies focus on?*

RQ-3.3: *What are the main aspects that the research questions deal with?*

RQ-4: *How do the approaches handle the fault locating process?* **Rationale:** this question treats issues related to how methods reach consensus on suspicious software elements, such as the evaluation functions, and search spaces. To this end, we subdivided this question into three sub-questions as follows.

RQ-4.1: *What are the meta-heuristics used?*

RQ-4.2: *How do the approaches handle the guidance of the search process?*

RQ-4.3: *What classes of search spaces are probed in SBFL studies?*

RQ-5: *How are the approaches evaluated?* **Rationale:** this question analyses the applicability of SBFL methods that include real cases. It helps the researcher to plan the evaluation of his/her methods and provides a basis for comparison in the research area. To address these issues, three sub-questions were considered:

RQ-5.1: *What baselines are used when evaluating SBFL methods?*

RQ-5.2: *What evaluation metrics are used when evaluating SBFL methods?*

RQ-5.3: *What benchmarks are used when evaluating SBFL methods?*

4. Search for primary studies

When searching for relevant studies, we follow the argumentation of Wohlin *et al.* [22] for achieving a good sample instead of exhaustively finding all primary studies. The systematic choice of the sources (*e.g.* quality indexed databases) and the scanning methods (*e.g.* application of search strings) promotes a better representation of the population for the targeted topic.

The following databases were elected, as recommended by Kitchenham *et al.* [6] and Petersen *et al.* [20]:

- IEEExplore (*http://ieeexplore.ieee.org*);
- ACM Digital Library (*http://dl.acm.org*);
- SCOPUS (*http://www.scopus.com*);
- Science Direct (*http://www.sciencedirect.com*);
- Engineering Village (*http://www.engineeringvillage.com*).

The search stage identifies papers using search strings in all databases that are relevant to the research field and keywords from the research questions should be the basis for formulating the start set, and are essential when searching for the initial set of papers to start the snowballing [20]. Our search string was built with sets of keywords so that papers have to match at least one keyword in each set to be selected.

The first set is composed of the word "localization" and its language variations and synonyms *e.g. localisation, locating, localising and localising.* The second set is composed of the word "fault" and the words that have been used as the same meaning in the research field such as "bug", "defect" and "error". The third set of keywords have the term "*search-based*", which indicates the solution strategy to the problem. This set look for papers that apply metaheuristic optimisation techniques to solve the fault localisation problem. The keyword "*metaheuristic*" along with its variations was also added to the set. Moreover, not every paper uses such words, instead some may prefer explicitly to use the name of the metaheuristic techniques applied. Hence, keywords associated with the most popular techniques applied to search-based software engineering should be selected. Such meta-heuristics include: *Hill Climbing*, *Simulated Annealing* and *Genetic Algorithms* [23]. Moreover, we also added keywords used in fault localisation surveys [4], and used in related areas such as search-based test case generation [24], they include: *Tabu Search*, *Ant Colony Optimisation*, *Genetic Programming* and *Particle Swarm Optimisation*.

Through pilot searches a significant volume of publications not related to software engineering was found; for instance, papers that treat fault localisation in other contexts, like automotive engineering [25] or electromagnetic wave propagation [26]. Hence, a set of keywords was added to the search string to restrict the results to publications on software engineering; *i.e.* to reduce the noise due to a number of non-relevant articles in this mapping study. As the terms “software engineering” or “software” might not be present in every paper, the following keyword set was established: “software” and “program”.

Since the search string screens papers that have at least one keyword from each set, the final search string was built with the logic operators *AND* and *OR* as follows:

(“software” OR “program”) AND (“bug” OR “defect” OR “fault” OR “error”) AND (“localization” OR “localisation” OR “locating” OR “localizing” OR “localising”) AND (“search based” OR “search-based” OR “search algorithm” OR “search-algorithm” OR “metaheuristic” OR “metaheuristics” OR “meta-heuristic” OR “meta-heuristics” OR “genetic” OR “evolutionary” OR “hill climbing” OR “hill climb” OR “annealing” OR “tabu” OR “colony” OR “swarm”)

Once the search string was built, the meta-data to be used on the search engines of the selected databases were defined. In this study the search is applied to *title*, *abstract* and *keywords*.

To deal with validity issues the quality improvement was conducted in two ways:

- **Independent assessment of the authors**. The first author tailored the search string according to the search engine’s features and carried out the search over the selected databases. To evaluate the study identification process the other authors created their own set of search terms and applied them to the search engines. The minor differences between the sets of studies obtained from the authors’ searches were settled by considering all studies present in both sets.

- **Inclusion of *control papers***. We developed the search string after performing a number of pilot searches to get relevant studies. The search string was evolved until the search results included a small set of papers that we expected to find as to they were flagged as relevant according to the authors’ perceptions:

  - S. Wang, D. Lo, L. Jiang, Lucia, H. Lau, Search-based fault localization, in: 2011 26th IEEE/ACM International Conference on Automated Software Engineering, ASE 2011, Proceedings, 2011.
  - L. Naish, Neelofar, K. Ramamohanarao, Multiple bug spectral fault localization using genetic programming, in: 2015 24th Australasian Software Engineering Conference, 2015.
  - S. Yoo, X. Xie, F. C. Kuo, T. Y. Chen, M. Harman, Human competitiveness of genetic programming in spectrum-based fault localisation: Theoretical and empirical analysis, ACM Transactions on Software Engineering and Methodology, 2017.

The reference period for the database search process was defined based on *the SBSE manifest* of Harman *et al.* as it was a milestone for the research area

[16, 27]. In this way, we set 2001 as the initial year for the search, and 2019 as the final year for the search (the last search was carried out on September 16th). Table 3a shows the number of papers found in the selected databases. The last row refers to the total number of primary studies obtained excluding any overlaps between the sources' results [1].

4.1. Inclusion and exclusion of studies: scoping the mapping

In addition to the search string, the inclusion and exclusion criteria define the mapping scope. This subsection explains the scope of the systematic mapping and justifies the exclusion criteria.

The present mapping focuses on the development and evaluation of fault localisation methods and techniques that apply metaheuristic search. To achieve this goal, we identify a set of *target scopes* that together define the mapping and describe the research field.

The following topics are presented below along with the rationale for each of them.

- **Software Engineering Topic**
  - *Scope*: The objective Software Engineering topic refers to the debugging of software faults.
  - *Rationale*: Studies should not emphasise topics other than software debugging or on other engineering areas. An out of scope example is: a method performs distributed localisation algorithms for wireless networks (*e.g.* [28]).
- **Debugging Focus**
  - *Scope*: The debugging focus refers to fault localisation techniques with respect to proposals of new methods as well as improvement of existing methods.
  - *Rationale*: Studies should focus on the development of fault localisation methods, instead of just using existing ones without aggregating improvements on them. Two out of scope examples are: the fitness function of a new approach uses an existing fault localisation method to guide the search for test data, in other words that refers to a test data generation approach (*e.g.* [29]), and fault localisation of Simulink models by generating test cases (*e.g.* [30]).
- **Algorithm**
  - *Scope*: The algorithm carries out metaheuristic search to solve the fault localisation problem.

[1] Search string variations were required for each database. They will be available after acceptance, as well as, all the raw data regarding this map

  - *Rationale*: Studies should deal with the fault localisation problem as a search problem and apply metaheuristic algorithms to solve that. An out of scope example is: the approach applies integer linear programming to break down the localisation problem into several smaller ones that can be dealt with independently (*e.g.* [31]).

- **Optimisation Target**
  - *Scope*: The search process scans somehow program elements in order to identify potential fault locations.
  - *Rationale*: Studies should carry out the metaheuristic search over the fault localisation search space itself instead of optimising another technique. An out of scope example is: a VSM (Vector Space Model) approach locates faults by using a Genetic Algorithm to configure the number of abstraction level topics for VSM (*e.g.* [32]).

- **Domain**
  - *Scope*: The faults to be located refers to functional faults in software written in general-purpose programming languages and on a broad application domain.
  - *Rationale*: Studies should cover fault localisation methods on a broad domain basis without being specific on programming language, application field, non-functional fault type, etc. Two out of scope examples are: faults appearing as the result of dynamic reconfigurations of a system due to context changes in a DSL (Domain Specific Language, *e.g.* [33]), and fault localisation of Simulink models by generating test cases (*e.g.* [30]).

- **Threat**
  - *Scope*: The research is written in an easy-to-read language and has been evaluated by the scientific community.
  - *Rationale*: Studies should be validated by peer-review process and be written in language of general scientific acceptance. An out of scope example is: a doctoral dissertation written in French (*e.g.* [34]).

In order to perform the scope of the mapping, nine exclusion criteria (EC) and two inclusion criteria (IC) were defined as follows.

EC1: Papers that do not use meta-heuristics in the problem solution, such as papers that apply only exact methods or do not treat the problem as an optimisation one.

EC2: Papers related to software fault localisation in specific domains (*e.g.* education), or focused on a particular programming language.

EC3: Papers focused on non-functional software failure (*e.g.* security vulnerability, electrical grids, physical systems);

EC4: Papers focused on the optimisation of test suites.

EC5: Papers focused on automatic fault repair.

EC6: Papers in which the optimisation algorithm is applied to a machine learning process instead of the fault localisation problem itself (*e.g.* optimising of k-means clustering or neural network).

EC7: Papers not related to software debugging.

EC8: Papers that are not written in English.

EC9: Papers not submitted to a scientific peer-review process, such as technical reports, technical notes, books, book chapters and websites.

IC1: Papers that treat the software fault localisation problem itself as a search-based optimisation problem.

IC2: Papers that support or enhance the SBFL methods, such as generation of fault localisation measures, or evaluation.

On the validity efforts, the two first authors independently evaluated the papers using three grades matching the selection criteria: met, possibly, or not met. Then the research expertise in the area of all the authors was used to decide the selection criteria for each paper.

The decision of how papers are evaluated in the selection was taken based on title, keywords, abstracts and optionally partial reading (*e.g.* introduction and conclusions) or full reading to dismiss any doubt. Table 3b shows the number of included and excluded papers per criterion. The table also shows the matching of papers over the databases.

Table 3: Summary of the search process.

(a) Number of studies obtained per database.

Source	Papers
IEEExplore (IEX)	54
ACM Digital Library (ACM)	51
SCOPUS (SCP)	145
Science Direct (SD)	40
Engineering Village (EV)	107
Total (duplicates excluded)	**233**

(b) Analysis of studies per criterion[a]

		Databases				
Criterion	Papers	SD	EV	SCP	ACM	IEX
IC1	**1**	1	1	1	0	0
IC2	**12**	0	10	10	4	4
EC1	**24**	2	14	16	5	8
EC2	**7**	0	3	6	2	1
EC3	**12**	4	6	7	0	5
EC4	**15**	0	10	13	2	5
EC5	**33**	2	16	21	13	6
EC6	**10**	0	7	8	2	4
EC7	**118**	31	39	63	23	22
EC8	**6**	0	6	5	0	0
EC9	**6**	2	3	3	0	0

[a] Some studies were classified in more than one exclusion criterion (EC).

To clarify the scope limits of this systematic mapping, Figure 2 presents the relation of each exclusion criterion with the target scopes aforementioned.

Edges between target scopes represent a coverage relation seen that *Software Engineering Topic* encompasses *Debugging Focus* (Fault Localisation, Software Testing, and Program Repair), which in its turn hold *Algorithm* and *Domain*, the same interpretation goes for the remains target scopes.

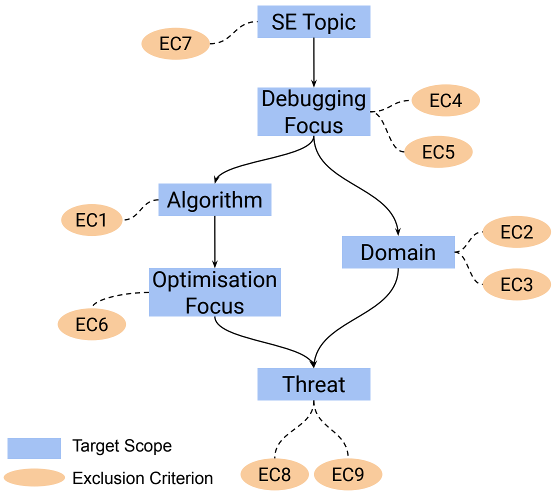


Figure 2: Relation between target scopes and exclusion criteria.

Target scopes compose the boundary of the mapping, so the exclusion criteria are used to apply such limits in the set of the found studies. For instance, EC4 and EC5 are used to exclude studies that although presenting research in *Software Engineering Topic* (Figure 2), are focused on improving test suites and automated repair programs, respectively. We utilise the target scope *Threat* to prevent the analysis of studies that represent potential threats to the results, even if they are perfectly satisfied by the other target scopes.

Figure 3 presents the number of excluded papers at the end of the selection of relevant primary studies, according to the target scope. The majority of excluded papers address an engineering topic distinct from software debugging (and software engineering). *Debugging focus* is the second reason for exclusion, followed by *Algorithm*, *Domain*, *Threat* and *Optimisation Target*.

The present mapping aims to cover papers that develop fault localisation techniques and methods based on metaheuristic search. Primary studies excluded due to *Debugging Focus* do not develop or improve fault localisation techniques and methods, but use existing ones to promote software testing and repair approaches (*e.g.* [29, 30]). A secondary study that falls in the intersection of Fault Localisation (FL) and Search based Software Engineering (SBSE) will potentially include such studies, but that focus is distinct from the present systematic mapping.

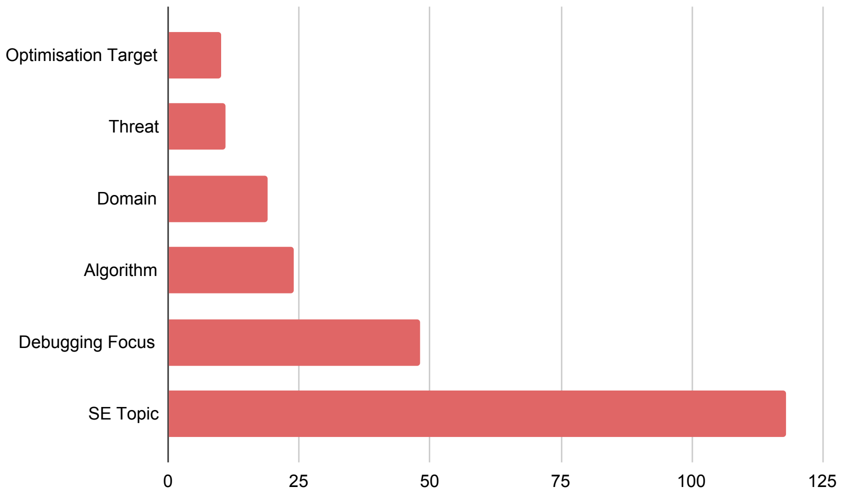


Figure 3: Excluded papers per target scope.

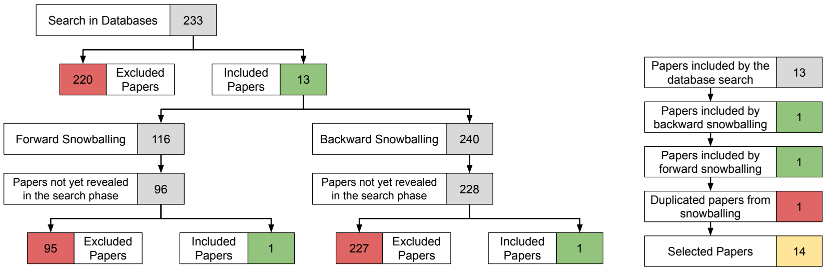
4.2. Snowballing

Harrold *et al.* [10] have advocated the use of snowballing as the main method to find relevant literature. In their recommendation, they highlight two processes: backward snowballing (BS) to search for new primary studies from the included papers' reference lists, and forward snowballing (FS) to search for new primary studies from the citations to the included papers. Both processes were carried out by the two first authors so that they could perform cross validation. Each resulting paper was analysed according to its title and abstract.

The BS considered all references cited by the papers included previously. The FS used the Google Scholar search engine due to its great capability of searching citations across many publishers available [35]. As a consequence the BS and FS processes found 228 and 96 papers, respectively, that means such papers had not been revealed in the database search phase. Then inclusion and exclusion criteria were applied to these papers, so BS and FS added the same paper to the included set of papers. After new analysis such paper did not present new references or new citations since all of them were already analysed previously, so new backward and forward snowballing cycles were not required.

4.3. Search Summary

Figures 4a and 4b summarise the search process. Table 4 lists in chronological order all papers selected after the application of the inclusion and exclusion criteria. Column ID identifies the relevant papers for the systematic mapping study from this point forward and follows the pattern Rx: R refers to the the adjective *relevant*, and x states a numerical sequence that results in $R1$, $R2$, and so on.


Search in Databases
233
220
Excluded Papers
Included Papers
13
Forward Snowballing
116
Backward Snowballing
240
Papers not yet revealed in the search phase
96
Papers not yet revealed in the search phase
228
95
Excluded Papers
Included Papers
1
227
Excluded Papers
Included Papers
1
Papers included by the database search
13
Papers included by backward snowballing
1
Papers included by forward snowballing
1
Duplicated papers from snowballing
1
Selected Papers
14

(a) The database search followed by the forward and backward snowballing, where both latter were carried out in parallel.

(b) Summary of the search process.

Figure 4: Numbers of the search process: grey, red, green and yellow boxes reveal obtained papers, excluded papers, included papers and final papers, respectively.

Table 4: Set of selected primary studies.

ID	Title	Year
[R1]	Search-based fault localization	2011
[R2]	Evolving human competitive spectra-based fault localisation techniques	2012
[R3]	Multiple Bug Spectral Fault Localization Using Genetic Programming	2015
[R4]	Empirical Evaluation of Conditional Operators in GP Based Fault Localization	2017
[R5]	FLUCCS: Using Code and Change Metrics to Improve Fault Localization	2017
[R6]	Genetic Programming-based Composition of Fault Localization Heuristics	2017
[R7]	Human Competitiveness of Genetic Programming in Spectrum-Based Fault Localisation: Theoretical and Empirical Analysis	2017
[R8]	Evolutionary Composition of Customised Fault Localisation Heuristics	2018
[R9]	Learning fault localisation for both humans and machines using multi-objective GP	2018
[R10]	Learning without peeking: Secure multi-party computation genetic programming	2018
[R11]	Localizing multiple software faults based on evolution algorithm	2018
[R12]	Mutation-Based Evolutionary Fault Localisation	2018
[R13]	Spectral-based fault localization using hyperbolic function	2018
[R14]	Empirical Evaluation of Fault Localisation Using Code and Change Metrics	2019

5. Data Extraction, Classification and Visualisation

The relevant studies were read in detail to extract the data needed to answer the research questions. The two first authors read fully all the primary studies, and both independently analysed and extracted data. The full-text analysis included annotating a digital version of each paper by using colours and adding comments so that each colour was related to a research question (*e.g.* brown for the benchmarks, orange for evaluation measures, and so on). The analyses were compared and disagreements were resolved by consensus or by consultation with the other authors who are experts in the study domain.

Threats in interpreting the data include researcher bias. To reduce these validity threats and gain confidence in the results, the other authors have checked the outcome. The colours used and the annotations promoted higher productivity and clarified the validation. Further analysis and consensus meetings were added to resolve disagreements and uncertainty.

The following subsections categorise the relevant studies and map the extracted data in order to respond the research questions by including charts and diagrams.

5.1. *How has the number and the frequency of publications evolved over the years?* [RQ-1]

Figure 5 depicts the number of SBFL papers over the years, and the areas in the graph show the publications in journals and conferences. The seminal study was published in 2011 by Wang *et al.*, and the authors called this approach "*search-based fault localisation*", as the FL heuristic composition problem was treated as a search problem for the first time. Most articles (78.6%) were produced from 2017, which shows the current interest in this area. Considering that more papers can be indexed by the search databases in 2019, we can see an increasing number of published papers in the last years and that search-based fault localisation keeps raising researchers' interest. We conjecture the research community have recognised the potential of the SBFL area to contribute to the automation of the FL efforts.

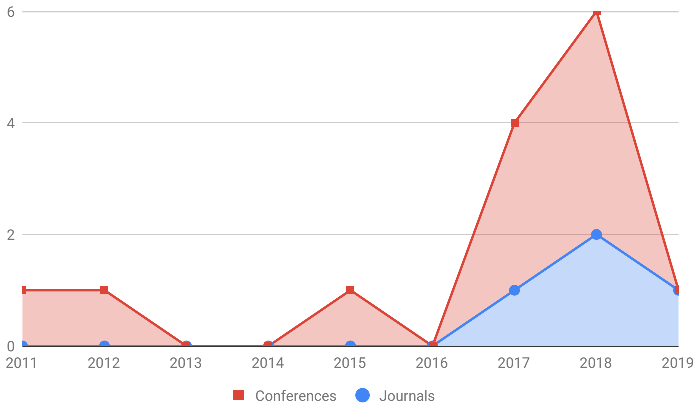


Figure 5: Number of papers over the years.

5.2. *What venues has the research on SBFL methods been published in?* [RQ-2]

The number of the studies per venue (journals, as well as peer-reviewed conferences) is shown in Table 5. We found 12 different publication venues. The publication fora is mainly from the areas of Software Engineering and Computational Intelligence. The first papers were published in conferences (2011-2015) but this changed to journals recently (2017-2019). The most preferred event is Symposium on Search Based Software Engineering (SSBSE) with 3 papers. There is a preference for conferences (10 out of 14) that include several valued by the research community such as the International Conference on Automated Software Engineering (ASE), Genetic and Evolutionary Computation Conference (GECCO), International Symposium on Software Testing and Analysis (ISSTA), and IEEE Congress on Evolutionary Computation (CEC). Regarding journals, the venues include ACM Transactions on Software Engineering and

Methodology, Journal of Systems and Software, Software - Practice and Experience and IEEE Transactions on Software Engineering. Overall, this indicates that SBFL studies are regarded as valuable scientific contributions, given that they have been published in quality forums.

Table 5: Publication venues.

Forum	**Papers**	**Years**
Conferences		
IEEE/ACM International Conference on Automated Software Engineering	1	2011
Symposium on Search-Based Software Engineering	3	2012, 2018(2)
Australasian Software Engineering Conference	1	2015
ACM SIGSOFT International Symposium on Software Testing and Analysis	1	2017
Genetic and Evolutionary Computation Conference	1	2017
Brazilian Workshop on Search based Software Engineering	1	2017
European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning	1	2018
IEEE Congress on Evolutionary Computation	1	2018
Journals		
ACM Transactions on Software Engineering and Methodology	1	2017
Journal of Systems and Software	1	2018
Software - Practice and Experience	1	2018
IEEE Transactions on Software Engineering	1	2019

5.3. ***What investigations and data are addressed by the studies?*** *[RQ-3]*

This section refers to the investigations and sources of fault data processed in the research field. The following sub-questions go through such aspects.

5.3.1. ***What data are considered as sources of faults to be located?*** *[RQ-3.1]*

On designing fault location methods, we consider data which are used as sources of information about existing software faults. Basically the research field deals with the following source categories:

- *Code coverage spectra* refers to the code covered by test cases. The execution of a test case produces a link between the executed code and the test result, *i.e.* whether or not the exercised control flow generates the expected result by the test case. In this sense, each software element e (*e.g.* a program statement) is associated the following variables: $e_p(e)$ ($e_f(e)$) is the number of pass (fail) program runs that execute element e, and $n_p(e)$ ($n_f(e)$) is the number of pass (fail) runs that do not execute element e;

- *Change and code metrics* refers to static information such as change frequency of program elements, how long a given program element has existed in the code base; lines of code (LOC), number of local variables, among others;

- *Mutation spectra* refers to the use of mutation analysis data: a set of program mutants are generated, each one differs from the program under the fault localisation process by a syntactical change in a statement. The idea is to assign suspiciousness to injected mutants, based on the assumption that test cases that kill mutants (*i.e.* make the mutants behaviour distinct from the original program) carry diagnostic power: the more often a statement affects failing tests, and the less often it affects passing tests, the more suspicious the statement is considered [36].

Table 6 details the data which are used as sources of faults in each study. The main source of fault information is *code coverage spectra*, since all the studies are grounded on it: seven of them are directly formulated from *code coverage spectra* variables (e_p, e_f, n_p, n_f) and the others are *code coverage-based metrics* that means they are also formulated from such variables; *e.g.* *Jaccard* and *Information Gain* in [R1, R6], *Ochiai* and *Tarantula* in [R4, R5, R6, R9, R10].

Table 6: Sources of faults information over the studies.

Fault data source	**Studies**
Code coverage spectra	[R2] [R3] [R7] [R8] [R11] [R12] [R13]
Code coverage-based metrics	[R1] [R4] [R5] [R6] [R8] [R9] [R10] [R14]
Code and change metrics	[R4] [R5] [R9] [R10] [R14]
Mutation spectra	[R12]

In Figure 6, which presents the preferences (number of papers) related to data sources, *code coverage spectra* is used in 100% of the studies and 43% of them (6 out of 14) are hybrid approaches as they also employ other fault sources: five studies use *code and change metrics* and one applied *mutation spectra*.

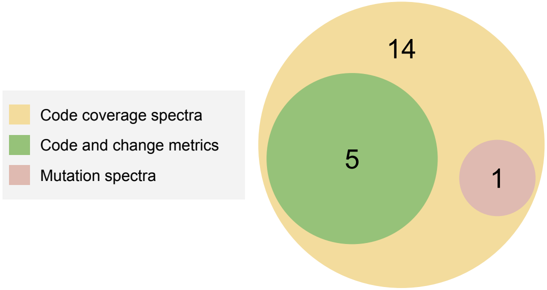


Figure 6: Preferences of data source.

5.3.2. *What do primary studies focus on? [RQ-3.2]*

In general, the primary studies deal with methods that assign suspiciousness scores to program elements, in order to set higher scores for faulty program elements related to the non-faulty ones. Thus such methods aim to place the faulty elements at the top when ranked by them.

Basically the research is divided into three categories that the studies fall into:

1. the proposition of a particular formula to calculate the suspiciousness of program elements as defective;

2. the generation of suspiciousness formulae by an optimisation algorithm;

3. the introduction of a method that ranks program elements as defective directly from an optimisation algorithm (without applying formulae to compute suspiciousness scores).

Figure 7 presents the distribution of papers over the categories of methods. *Formulae generation* is the preferred in the research field as it cover 71% of the studies. The two first categories refer to *formula-based methods* because the suspiciousness ranking is created from formulae evolved by optimisation algorithms (from the scores obtained from the calculation of formulae). The third category refers to *direct-ranking methods* since the optimisation process itself creates the suspiciousness ranking. Note that most of the studies (13 out of 14) cope with *formula-based methods* and this exposes a research gap to exploit *direct-ranking methods*.

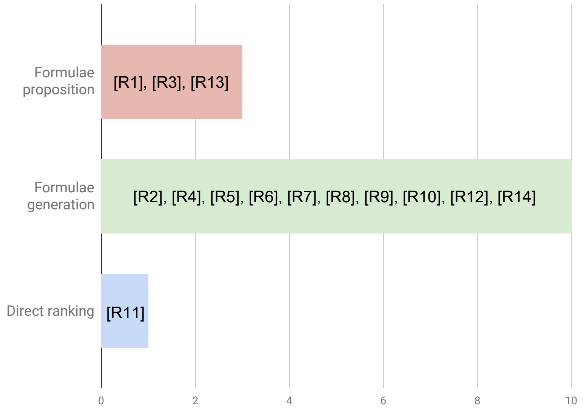


Figure 7: Method categories in the research field.

In the first category, a formula is proposed in [R1] that represents a composition of 22 previous formulae. Another formula is proposed in [R3] whose variables are code coverage spectra data. In both studies the formulae have a set of weights (*e.g.* w_1, w_2, K_1, K_2) whose values must be determined by optimisation algorithms. The latter was extended in [R13] by aggregating a pruning technique aiming to decrease the size of training data (and the learning time), larger data sets in experiments, and an efficiency analysis.

Most of the studies refer to the generation of suspiciousness formulae: basically optimisation algorithms compose formulae from a set of variables (sources of fault data) and mathematical operators, and evolve such formulae according to a fitness function. These studies share the belief that the machine-evolved formulae are at least as good as the ones designed by humans. The seminal study was published in 2012 [R2], and the idea has received improvements and extensions since then: additional fault data source [R5, R12, R14]; more robust evaluation of generated formulae [R7]; a ternary conditional operator to compose formulae [R4]; a multi-objective approach to achieve higher scores for defective elements against lower scores for non-defective ones [R9]; the use of previous heuristics for calculating suspiciousness scores (*i.e.*, human-proposed equations to rank program elements) which are used as variables for the generation of new formulas [R5, R6, R8, R14]; a secure multiparty mechanism so that one party learns a model of training data provided by another party but keeping the inputs hidden from each other [R10].

Despite the preference for *formula-based methods* in this research area, Zheng *et al.* [R11] proposed a method to derive a suspiciousness ranking directly from the optimisation process, by evaluating the combinations of program entities as candidates with faults, including dealing with multi-fault programs, instead of considering statements in isolation.

5.3.3. ***What are the main aspects that the research questions deal with?*** *[RQ-3.3]*

According to Creswell [37], a Research Question (RQ) articulates "the uncertainty that the investigator wants to resolve by performing his/her study". We find as a consequence of the authors' methodological style, some studies do not explicitly present the RQs in the text itself, so we extract them from 'the uncertainty being resolved' whose content is mainly located in the papers' results section. Overall the studies measure the method's execution behaviour and compare the results with those from other methods in order to gather data (results) that support the answer to the research questions.

Figure 8 presents the focus of the research questions in the studies. Basically the research questions deal with *efficacy* and *efficiency* as well as *other issues.* On using the terms efficacy and efficiency the papers do not have consensus on their meaning so we take efficacy to mean *how effective for fault localisation the method is* (*e.g.* the suspiciousness rank of a faulty program element), and efficiency to mean *how much effort the method demands* (*e.g.* the execution time of the method).

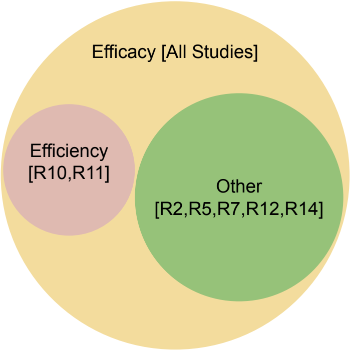


Figure 8: Focus of research questions in SBFL studies.

The main concern over the research field is the effectiveness when locating faults, so all the studies touch on functional aspects towards investigating the *efficacy* as shown by the Venn Diagram in Figure 8. Furthermore five (out of 14) studies approach *other issues* that are closely related to efficacy such as: 'what design insights can be learned from the Genetic Programming-evolved formulae?'[R2]; 'how much do the code and change metrics contribute to the fault localization?'[R5]; 'is there a greatest formula that performs best in all existing programs and faults?'[R7]; 'how does mutation spectra quality impact fault localisation ability?'[R12]; and 'does the choice of learning algorithm affect the effectiveness?'[R14].

Two studies cope with non-functional aspects and they have included the time spent on configuration and evolution of the optimisation algorithm in their analysis. Kim *et al.* [R10] concluded their method can be up to three orders of magnitude slower than the baselines, despite it being competitive on functional issues. Zheng *et al.* [R11] did not perform comparisons against the time spent on baseline methods, but what is "acceptable spent time in the real development environment" (*e.g.* 23 seconds for the worst scenario of benchmark programs).

5.4. ***How do the approaches handle the fault locating process?*** *[RQ-4]*

This section deals with how methods achieve the decision on suspicious software elements. The following sub-questions go through such aspects.

5.4.1. ***What are the meta-heuristics used?*** *[RQ-4.1]*

Table 7 shows the meta-heuristics addressed per primary study, while presenting the algorithm distributions in the SBFL area.

Table 7: Algorithms per SBFL study.

Algorithm	**Study**
Genetic programming	[R2, R3, R4, R5, R6, R7, R8, R10, R12, R13, R14]
Simulated annealing	[R11]
Genetic algorithm	[R1, R11]
NSGA-II	[R9]

Genetic Programming (GP) dominates the others (11 out of 14 studies), mainly due to its way of generating rules: suspiciousness formulae in the fault localisation context. *Simulated Annealing* and *Genetic Algorithm (GA)* are used in one and two studies respectively. *Non-dominated Sorting Genetic Algorithm II (NSGA-II)* is employed in [R9], that is a multi-objective version of a prior study ([R5]). Moreover one study uses more then one algorithm: *simulated annealing* optimises a *genetic algorithm* based solution in [R11].

5.4.2. ***How do the approaches handle the guidance of the search process?*** *[RQ-4.2]*

Overall the guidance of a search process is grounded on an objective function aiming to drive the process to achieve better solutions. According to Figure 9, the preferred metric used as fitness function is *mean expense* (13 out of 14 studies) followed by *weighted coverage on failed test cases*, the latter is applied only in [R11]. Despite its predilection as a fitness function, *Mean expense* produces measures that are related to the percentage of examined code to find faults with respect to the code size (*e.g.* number of lines of executable code). For this reason its use is questioned by test engineers [38] as these metrics are related to code size rather than being an absolute value such as number of statements inspected until faults are found. On the other hand the research in [R11] differs from the others as it proposes a new fitness function (formula) based on the importance of a program element related to its coverage on failed test cases in a multi-fault scenario.

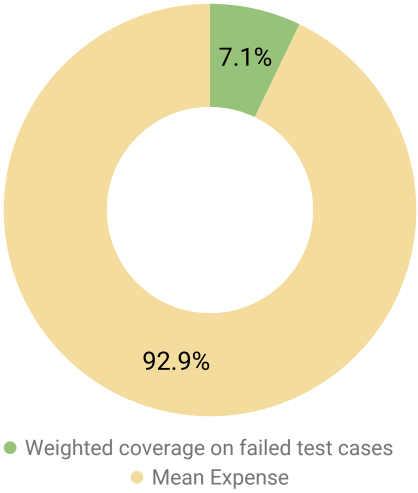


Figure 9: Fitness functions in the research area.

5.4.3. ***What classes of search spaces are probed in SBFL studies?*** *[RQ-4.3]*

Search Space supports the way to model the set of all potential solutions for a problem, so that each element in the set represents one feasible solution. Table 8 shows the studies fall into three categories as follows:

- *All compositions of weights of a suspiciousness formula* reports on the *weights* used in an equation whose values are obtained from an optimisation algorithm;

- *All valid suspiciousness formulae* refers to the valid suspiciousness equations obtained from evolutionary approaches;

- *All suspiciousness sequences of program elements* denotes all possible rankings of the most likely elements to be defective.

Table 8: Search space over SBFL studies.

Search space	**Studies**
All valid suspiciousness formulae	[R2, R4, R5, R6, R7, R8, R9, R10, R12, R14]
All compositions of weights k_1, k_2, k_3	[R3, R13]
All compositions of weights $w_1, w_2, ..., w_{22}$	[R1]
All suspiciousness sequences of program elements	[R11]

On *all compositions of weights of a suspiciousness formula*, weights k_1, k_2, k_3 in [R3, R13] have the domain $k_i \in R \mid 0 \leq \{k_1, k_2\} \leq 100 \wedge 0 \leq k_3 \leq 2$, and weights $w_1, w_2, ..., w_{22}$ in [R1] have the domain $w_i \in R \mid 0 \leq w_i \leq 1$. In both cases such domains are exploited by the optimisation algorithms selected by the authors.

Regarding *all valid suspiciousness formulae*, the studies aim to generate better formulae to identify the faulty program elements by evolving equations composed by operators (*e.g.* basic math operations) and operands (sources of fault data); for instance, code coverage spectra variables in [R2, R3]; code and change metrics in [R5, R4]; mutation spectra variables in [R12]. Notwithstanding the results obtained from evolved equations the literature states there is no single optimal formula that performs best for efficacy over all contexts [R7].

With respect to *all suspiciousness sequences of program elements* the search space addresses possible sequences of suspicious elements from the entire set of program elements. Due to the huge solutions domain Zheng *et al.* restrict the candidates to ones covered by all the failed test cases aiming to reduce the search space [R11].

The distribution in Table 8 reveals most of the studies (71.4%) cope with the search space of *all valid suspiciousness formulae*, 21.5% evolve solutions from *all compositions of weights of a suspiciousness formula*, and 7.1% exploit *all suspiciousness sequences of program elements.* These classes clarify how the studies model the search-based problem that defines its search space.

5.5. ***How are the approaches evaluated?*** *[RQ-5]*

This section focuses on the way the research area carries out analyses that help the researcher to plan the evaluation of his/her approaches and constitutes a basis for comparison in the research area. The following sub-questions analyse this question.

5.5.1. ***What baselines are used when evaluating SBFL methods?*** *[RQ-5.1]*

Baselines refer to comparison references on evaluating new SBFL methods. We abstract from the studies the following baseline categories:

- *Analytical metric* refers to a prior metric that was not generated by an optimisation algorithm (*i.e.* proposed by humans).
- *GP-formula* is a specific FL formula previously generated by a genetic programming approach.

- *GP-method* denotes a genetic programming method that dynamically generates a solution at the evaluation time.

- *GA-method* denotes a genetic algorithm method that dynamically generates a solution at the evaluation time.

- *SA-method* denotes a simulated annealing method that dynamically generates a solution at the evaluation time.

- *Other* refers to solutions from methods that are not based on metaheuristic search, such as learning-to-rank and linear programming.

Table 9 presents the baseline distribution over the SBFL field, one can observe baseline categories per primary study. Observe that 11 out 14 of the studies use more than one category, *i.e. hybrid baseline*. Figure 10 depicts preferences related to the number of studies. Most of the studies (78.6%) use *analytical metrics* as baselines. *Metaheuristic method* encompasses 57.1% of studies, it merges the categories *GP method*, *GA method* and *SA method*, which means that the research field values metaheuristic-based methods as consistent baselines.

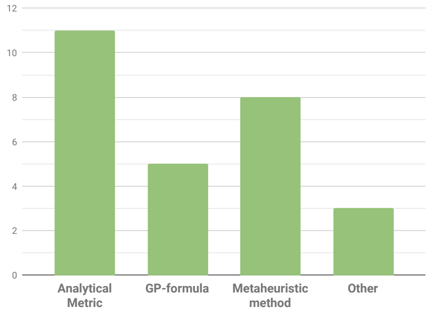


Figure 10: Baseline categories over SBFL field.

The use of analytical metrics per study is detailed as follows: Tarantula, Ochiai and Information Gain in [R1]; Tarantula, Ochiai, Jaccard, OP1, OP2, AMPLE, Wong1, Wong2 and Wong3 in [R2]; Tarantula, Ochiai, O^p, O^d, Zoltar and Kulczynski2 in [R3]; Ochiai, Jaccard, ER1a, ER1b, ER5a, ER5b and ER5c in [R5]; Tarantula, Ochiai, Jaccard, ϕ-Coefficient, Yule's Q, Yule's Y, Kappa, J-Measure, Gini Index, Support, Confidence, Laplace, Cosine, Piatetsky-Shapiro's, Certainty Factor, Added Value, Klosgen and Information Gain in [R6]; Naish1, Naish2, Wong1, Russel & Rao and Binary in [R7]; Tarantula, Ochiai, Ochiai2, Jaccard, Braun-Banquet, Dennis, Mountford, Fossum, Pearson, Gower, Michael, Pirce, Baroni-Urbani & Buser, Tarwid, Ample, Phi (Geometric Mean), Arithmetic Mean, Cohen, Fleiss, Zoltar, Harmonic Mean, Rogot2, Simple Matching, Rogers & Tanimoto, Hamming, Hamann, Sokal, Scott, Rogot1, Kulczynski, Anderberg, Dice, Goodman and Sorensen-Dice in [R8]; Tarantula, Ochiai, Jaccard,

Table 9: Baselines in SBFL studies.

Study	Analytical metric	GP-formula	GP-method	GA-method	SA-method	Other
[R1]	√	–	–	–	√	–
[R2]	√	–	–	–	–	–
[R3]	√	–	–	–	–	–
[R4]	–	–	√	–	–	–
[R5]	√	√	–	–	–	√
[R6]	√	–	–	√	–	–
[R7]	√	√	–	–	–	–
[R8]	√	–	–	√	–	–
[R9]	–	–	√	–	–	–
[R10]	–	–	√	–	–	–
[R11]	√	√	–	–	–	√
[R12]	√	–	√	–	–	–
[R13]	√	√	–	–	√	–
[R14]	√	√	–	–	–	√

OP2, Dstar and Ample in [R11]; Tarantula, Ochiai, OP2, Barinel and DStar in [R12]; Tarantula, Ochiai, O^p, O^d, Zoltar, Kulczynski1, Kulczynski2, Ample, Wong1 and Wong2 in [R13]; Ochiai, Jaccard, ER1a, ER1b, ER5a, ER5b and ER5c in [R14].

The baselines that fall into *GP-formula* category are: GP02, GP03 and GP19 in [R5, R7, R14]; and GP13 in [R5, R11, R13, R14]. Regarding the *Other* category, [R5] and [R14] compared the proposed GP-based solution against variants implemented with other learning algorithms such as Ranking Support Vector Machine, Random Forest and Gaussian Process Modelling. Moreover [R11] adopts a linear programming model to solve a multi-fault localisation problem.

5.5.2. ***What evaluation metrics are used when evaluating SBFL methods?*** *[RQ-5.2]*

When evaluating a SBFL method, evaluation metrics are generally applied to obtain the measures on the object of study such as ability to locate faults and execution cost. These measures are then compared with those obtained from baselines in order to analyse the method under evaluation. We identify six distinct evaluation metrics used by the SBFL studies. To normalise the mapping data, we adopt a unique definition for such metrics as follows:

- *Expense*: Proportion of inspected program elements, related to the total number of program elements, it takes to find the first fault. This is similar to the *EXAM* metric but these definitions have not been agreed in the literature (*e.g.* the definitions of *EXAM* in [36] and [39] are distinct).
- *Average rank percentage*: Proportion of examined code to locate all program faults [R13].
- *Accuracy*: The number of faults within the top-n elements in the suspiciousness rank [R4].

- *Wasted effort*: The number of program elements that need to be investigated in order to reach the fault [R4].
- *Mean average precision*: The mean of how precise the method is to locate faults on average, there is an example on how to apply this metric in [40].
- *Execution time*: Elapsed time in seconds.

Table 10 shows the studies that use each evaluation metric. Since an evaluation metric measures a particular perspective of SBFL methods, the table shows most of the studies (10 out of 14) use more than one metric and all metrics are applied by more than one study. In some cases the studies do not apply a particular metric in the same way. For instance, the *accuracy* metric is defined as the number of programs for which an algorithm ranks all faulty statements among the top-n positions in [R11], but this metric is defined as the number of faults within the top-n elements in [R4]. We conjecture that this variety impacts the comparison of results between studies as the manner in which a metric is defined can change the results. Apart from that the studies consider the measures over all faulty versions and over a number of executions (most use 30 runs) in order to deal with stochastic effects.

Table 10: Evaluation metrics over SBFL research area.

Evaluation Metric	**[R1]**	**[R2]**	**[R3]**	**[R4]**	**[R5]**	**[R6]**	**[R7]**	**[R8]**	**[R9]**	**[R10]**	**[R11]**	**[R12]**	**[R13]**	**[R14]**
Accuracy	–	–	–	√	√	√	–	√	√	–	√	√	–	√
Average rank percentage	√	–	√	–	–	–	–	–	–	–	√	–	√	–
Execution time	–	–	–	–	–	–	–	–	–	√	√	–	–	–
Expense	–	√	–	–	–	√	√	–	–	–	√	√	√	–
Mean Average Precision	–	–	–	√	√	–	–	–	–	–	–	–	–	√
Wasted Effort	–	–	–	√	√	√	–	√	√	√	–	√	–	√

Figure 11 depicts the evaluation metrics used over method categories. *Accuracy* and *wasted effort* are the most used metrics (each 57.1% of the studies) and the unique metric applied to evaluate methods in all categories is the *expense* metric. The most used metrics per category are *wasted effort* followed by *accuracy* and *expense*, all related to *formulae generation* category, respectively in 8, 7 and 4 studies. Moreover *formulae generation* encompasses almost all metrics used over the research field except *average rank percentage.* The less used metric is the one related to efficiency evaluation, *i.e. execution time.*

5.5.3. ***What benchmarks are used when evaluating SBFL methods?*** *[RQ-5.3]*

The following groups were abstracted from the benchmark programs used in the studies: (i) *siemens*, a set of seven C programs originally created to support controlled experimentation with testing techniques [41, 42]; (ii) *defects4j*, real programs written in Java for controlled testing studies [43]; (iii) *linux utilities*, linux programs written in the C language; (iv) *space*, an array definition language interpreter written in the C language [44]; (v) *codeflaws*, a collection of

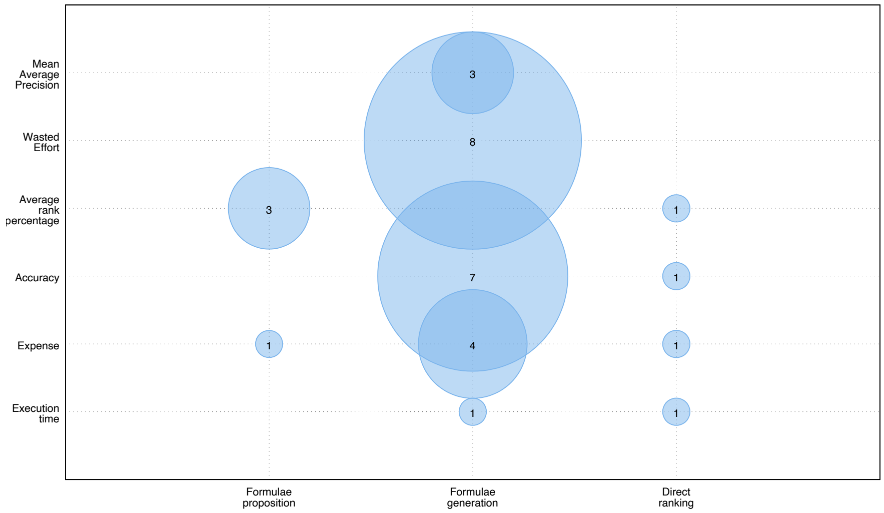


Figure 11: Number of papers in each methods categories that employ each evaluation metrics.

programs from a platform for programming contests [45]; (vi) *model programs*, small C programs with four statements designed for very controlled experiments; and (vii) *symbolic regression problems* described in [46].

The distribution of programs in the studies with respect to the first four groups is described as follows:

- *siemens*: print_tokens, replace and tot_info in [R1, R3, R6, R8, R11, R13]; print_tokens2 in [R1, R6, R8, R11, R12, R13]; schedule in [R1, R6, R8, R12, R13]; schedule2 in [R1, R6, R8]; tcas in [R1, R3, R6, R8, R12, R13];
- *defects4j*: lang, joda-time and closure in [R4, R5, R9, R11, R14]; math in [R4, R5, R9, R11, R12, R14]; mockito in [R10, R14]; chart in [R11, R14];
- *linux utilities*: grep, gzip and sed in [R2, R7, R11, R13]; flex in [R2, R7, R13]; cal, checkeq, col, spline, tr and uniq in [R3]; and
- *space*: [R7, R11, R13].

Table 11 shows how benchmarks are allocated over SBFL studies that include language, fault nature, and fault cardinality. Regarding programming language, C and Java are used in all studies but only two studies use both languages, and there is a slight preference for C programs (nine against seven studies). On fault cardinality, all studies exploit single fault benchmarks and most of them (9 out of 14) also exploit multiple fault benchmarks. Regarding fault nature, nine studies (64.3%) use artificial fault benchmarks and seven studies (50.0%) use real fault benchmarks. Only two studies ([R12] and [R11]) employ benchmarks that exploit both languages (C and Java), both fault cardinalities (single and multiple) as well as a combination of real and synthetic bugs. Furthermore

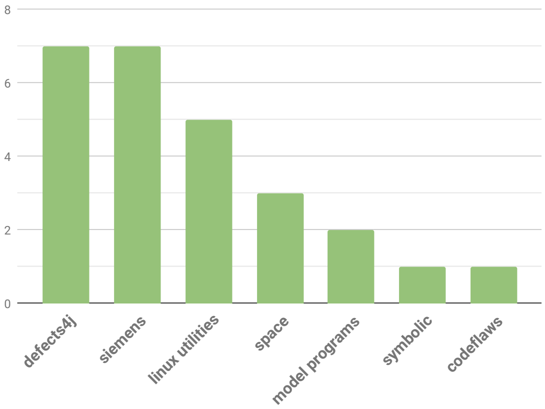


Figure 12: Benchmark preferences over SBFL research field.

Defects4j is the only benchmark written in Java and combines real and multiple fault contexts.

Figure 12 shows benchmark preferences over the SBFL research field. *Defects4j* and *siemens* are the most popular in the research field (seven studies each), followed by *linux programs* (five studies). These benchmarks as a whole dominate the research field, since they are present in all studies, either alone or in combination. Figure 13 shows the four most preferred benchmark groups over method categories. *Siemens*, *Linux utilities* and *space* are present in all method categories whist *defects4j* concentrates almost all of its use in the *formulae generation* category. The figure also shows that the *formulae generation* category uses all benchmark groups, that means this method category pays attention to more benchmark alternatives.

Table 11: Benchmarks over SBFL studies.

Studies	**Benchmarks**	**Language**		**Fault nature**		**Fault cardinality**	
		C	Java	Artificial	Real	Single	Multiple
[R1]	siemens	√	–	√	–	√	–
[R2]	linux utilities	√	–	√	–	√	–
[R3]	model programs, linux utilities, siemens	√	–	√	–	√	√
[R4]	defects4j	–	√	–	√	√	√
[R5]	defects4j	–	√	–	√	√	√
[R6]	siemens	√	–	√	–	√	–
[R7]	linux utilities, space	√	–	√	–	√	–
[R8]	siemens	√	–	√	–	√	–
[R9]	defects4j	–	√	–	√	√	√
[R10]	symbolic regression problems, defects4j	–	√	–	√	√	√
[R11]	siemens, defects4j, linux utilities, space	√	√	√	√	√	√
[R12]	siemens, codeflaws, defects4j	√	√	√	√	√	√
[R13]	model programs, linux utilities, siemens, space	√	–	√	–	√	√
[R14]	defects4j	–	√	–	√	√	√

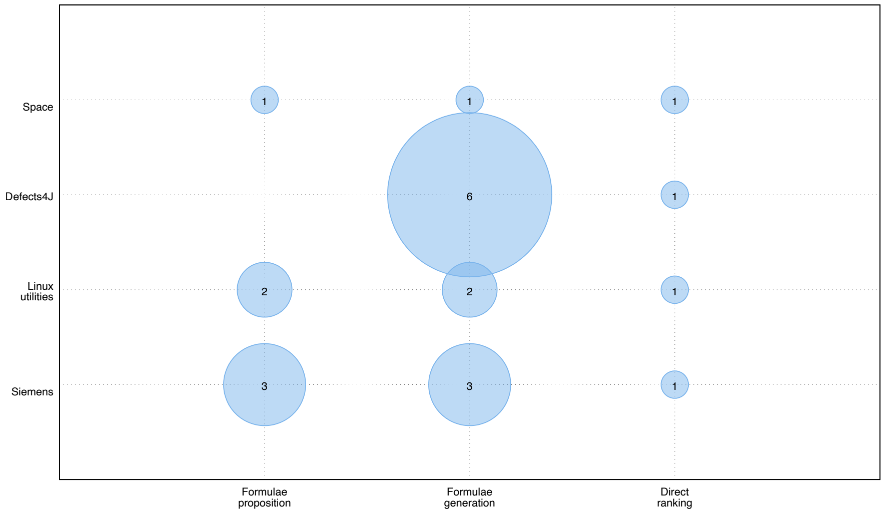


Figure 13: Benchmark preferences over SBFL method categories.

6. Summary of results and research opportunities

In this section, we present a synthesis of our findings regarding the SBLF field and identify research opportunities.

6.1. Data sources of software faults

The studies have covered the basis of the fault data sources on code coverage spectra, code and change metrics as well as mutation spectra. We found the predilection is coverage spectra as they were the basis for all studies; furthermore 43% of studies are hybrid approaches as they also employ other fault sources: five studies use *code and coverage metrics* and one applies *mutation spectra*. Based on the fact of the studies explore just three categories of fault sources, we conclude that to aggregate new sources of fault information such as data flow coverage, and to extend the investigation to code and change data as well as to mutation spectra is a research opportunity. Moreover as stated in [R12] it is relevant to analyse the quality of sources of fault data and how such quality can impact on the effectiveness of SBFL methods.

6.2. Main focuses of research area

Overall SBFL research deals with methods that assign suspiciousness scores to program elements, in order to set higher scores for faulty programs. Most of the studies (92.9%) refer to *formula-based methods* since the suspiciousness ranking is created from the execution of formulae that are evolved by optimisation algorithms. These studies share the belief that machine-evolved formulae are at least as good as the ones designed by humans [R7]. On the other hand the optimisation process itself of *direct-ranking methods* generates the suspiciousness ranking (7.1% of the studies) instead of using a ranking calculated from formulae. The present tendency of using *formula-based methods* reveals an open field to invest research efforts to develop *direct-ranking methods*; such research can also use new sources of fault data as described in Subsection 6.1.

6.3. Non-functional properties

All relevant studies describe their investigation focused on efficacy in the sense that empirical analysis measures the fault localisation ability of SBFL methods. It means researchers perceive that the methods must not fail at some level to locate software faults. Our mapping reveals that a minority of the studies also look at other aspects beyond efficacy such as the execution time of the method. This may impact our confidence when using and adapting of SBFL methods to the software engineers' expectations. Thus there is an important gap surrounding non-functional aspects of SBFL approaches as well as how such investigations can be evaluated, in order to reduce the burden of selecting the most appropriate solution based only on functional requirements. Moreover the choice of appropriate benchmarks is vital to the validity and contribution of the analysis of non-functional properties, for instance the computational time increases with executable LOC (Lines Of Code) and the number of test cases.

6.4. Metaheuristic algorithms

The algorithms used in SBFL studies are Genetic algorithm, Genetic Programming, Simulated annealing, and Non-dominated Sorting Genetic Algorithm (NSGA II). Genetic Programming dominates the others (11 out of 14 studies), mainly due to its ability to generate rules: suspiciousness formulae in the fault localisation context. There are many ways to explore the use of meta-heuristics, so researchers' choice of algorithms is also a research opportunity. Furthermore in addition to the used algorithms other metaheuristic algorithms fit with SBFL problem such as Hill Climbing, Bee Colony, PSO among others which also could be investigated.

6.5. Multi-objective approaches

The majority of the approaches use mono-objective algorithms. The unique method that focuses on multi-objective reasoning tries to achieve higher scores for defective elements against lower scores for non-defective ones [R9]. We conjecture that the use of many-objective optimisation algorithms should be further investigated in the SBFL context. For instance the effectiveness of SBFL methods depends on the training data so the selection of more robust benchmarks must be considered (larger programs, larger test case sets). In this sense multi-objective algorithms are fitter to cope with conflicting objectives; *e.g.* better method efficacy and shorter time execution.

6.6. Objective functions

Two objective functions were identified from this mapping study. The preferred one (*mean expense* in 92.9% of the studies) refers to the percentage of examined code to find faults with respect to the code size. On the other hand one study [R11] exploits the importance of a program element related to its coverage on failed test cases. Thus we point out a research opportunity to focus the objective function on *de facto* the number of the inspected program elements as it is preferred by software engineers rather than the percentage of examined code, since the latter depends on the program size. Furthermore the effectiveness of the function proposed in [R11] could be evaluated against the others.

6.7. Search spaces

Basically three categories of search space were identified: all valid compositions of weights that are actually a set of constants present in a suspiciousness formula; all valid suspiciousness formulae; and all suspiciousness sequences of program elements. The two first categories dominate as they take place in 13 out of 14 of the studies; they refer to indirect search spaces as the search itself occurs outside the code *i.e.* in spaces related to constructing formulas. The third refers to searching for solutions directly in the code. We presuppose it is a research opportunity to exploit on a larger scale the search spaces in the code, since the software faults themselves are located in the code. Moreover it can be profitable to deal with hybrid search spaces, for instance solutions of an indirect search space can be enriched by those of the direct search space and vice-verse.

6.8. Baselines

One finding is that the baselines fall into four categories: analytical metrics such as formulae designed by humans mainly based on their intuition; GP-formulas that were previously generated by a genetic programming approach; metaheuristic methods that dynamically generate a solution at the evaluation time; and others such as a linear programming model to solve a multi-fault localisation problem. Regarding solutions based on formulae, both evolved by machine and designed by humans, a prior finding is there is no formula that has been proved to be the best for all contexts [R7]. Most of the studies (78.6%) use analytical metrics, and metaheuristic methods encompass 57.1% of studies. We found that 11 out 14 of the studies use more than one category, *i.e.* hybrid baseline, and also all baselines categories are applied at least in three studies. This indicates the research area is aware of the four baseline categories. Further research is needed into using benchmarks to determine the ones most fit for classes of faults, programming language, and fault nature (synthetic and real).

6.9. Evaluation metrics

A variety of evaluation metrics (six metrics) are used in the research field. *Expense*, *accuracy* and *wasted effort* are the most used and show the main concern is the efficacy of SBFL methods. In some cases, the studies do not apply a particular metric in the same way. For instance, the *accuracy* metric is defined as the number of programs in which an algorithm ranks all faulty statements among the top-n positions as well as the number of faults within the top-n elements. We conjecture that this difference can impact the comparison of results between studies as the manner in which a metric is defined changes the results; *i.e.* the strength of evidence when considering a particular definition. We suggest a further investigation to map the ways an evaluation metric has been applied in the research field in order to determine how the results of a method compare.

6.10. Benchmarks

We abstracted seven groups of benchmarks over the papers: defects4j, siemens, linux utilities, space, model programs, symbolic regression problems, and codeflaws. Defects4j and siemens are the most popular in the research field (seven studies each), followed by linux programs (five studies); they dominate the research field, since they are present in all studies, either alone or in combination. C and Java are the most used languages but there is small preference for C programs (64.3% against 50.0%). Defects4j is the unique benchmark written in Java and combines real and multiple fault contexts. Finally, only 14.3% of the studies employ benchmarks that exploit both languages (C and Java), both fault cardinalities (single and multiple) as well as a combination of real and synthetic bugs. We suggest additional research to compare SBFL benchmarks and how fit they are to: SBFL methods, classes of fault, and evaluation metrics.

6.11. Secondary study

The present mapping aims to cover papers that develop fault localisation techniques based on metaheuristic search, that focus is distinct from the development of software testing and repair methods. A secondary study that falls in the intersection of Fault Localisation (FL) and Search based Software Engineering (SBSE) will potentially include studies that use existing FL techniques and methods to promote software testing and repair approaches. It is a finding and further investigation to conduct a secondary study that covers the intersection of FL and SBSE. With that important research questions will arise: How does test data generation impact FL ability? Which domains fit with each specific FL method? What benchmarks / baselines / evaluation metrics are used for FL in specific language / domain / algorithm / debugging combination?

7. Threats to Validity

We present some threats to the validity of our results by following the guidelines of Wohlin *et al.* [47].

Construct validity refers to the relation between the theory behind the experiment and the observation(s), i.e. what the researcher has in mind and what is investigated according to the research questions. Regarding the research questions, we defined them in discussion meetings to reach alignment with the goals of the mapping, so it became a mitigated threat. On the search string, it impacts the number of papers found and as a consequence the results. To mitigate such a threat, we created three groups of terms and refined them by performing pilot searches. Many search simulations were performed in order to cover the goals and research questions. Furthermore independent assessment of the authors and the inclusion of control papers took place to deal with threats. Related to the data sources, we carried out the search over databases which are well known sources in the literature. They returned studies that were published in conferences and journals of the research field, thus including the most relevant studies.

Internal validity copes with the relationship between the treatment and the output. Firstly we performed the study selection by defining and applying inclusion and exclusion criteria that demanded discussions and decisions in line with the mapping scope. As a study's omission is a threat in any mapping study, we carried out a rigorous searching and selection process and followed the argumentation of Wohlin *et al.* [22] for achieving a good sample instead of finding all primary studies. Subjectivity in the data extraction was an effort-demanding activity; the full-text analysis included annotating a digital version of each paper by using colours and adding comments. The two first authors applied the classification schema to perform all data collection from the relevant primary studies, then the other authors checked the outcome. We had many meetings and discussions during the selection and extraction of studies.

Threats to conclusion validity deal with the ability to reach and describe the correct conclusions from the study, and they are impacted by the classification scheme. We created the categories used in our analysis interactively. Basically we analysed which types of information were common or similar over the primary studies and used this to update the classification by abstracting relevant dimensions to answer the research questions. To deal with the validity, the classification schema have evolved throughout the authors' discussions until we reached the current version after common agreement. These threats occur in many classification schemes analysis.

8. Concluding remarks

Search-based fault localisation (SBFL) is the research field that deals with the use of optimisation techniques to automate, or partially automate the location of faulty code. As faults are constantly introduced and fixed during the software lifecycle and locating faults is a very time-consuming task, SBFL is an important research subject whose first research initiatives occurred in 2011 (according to our findings). We observed an increasing interest in the field since 2017, given by the increasing number of papers found in the last two years.

The mapping described in this paper adds value to the understanding of the SBFL field and is fundamental with respect to the status and the potential research opportunities. Formula-based methods dominate SBFL research and this means the fault suspiciousness ranking is mainly created from execution of formulae evolved by optimisation algorithms. Genetic Programming is the most used metaheuristic algorithm, mainly due to its ability to generate rules: suspiciousness formulae in the fault localisation context. The majority of the approaches use mono-objective reasoning, but one research paper applies a multi-objective algorithm.

The search process is mainly based on the percentage of examined code to find faults, with respect to the code size. On the sources of fault data the predilection is code coverage spectra as all studies are somehow grounded on that spectra, and over half of the studies also employ other fault sources. Overall the evaluations are focused on efficacy since the main concern is the fault localisation ability of SBFL methods. The major search space is all valid suspiciousness formulae composed by mathematical operators and operands related to the source of fault data. A variety of evaluation metrics are used in the research field but most of the methods are evaluated by measures such as expense, accuracy and wasted effort. However there is little consensus on how to apply the evaluation metrics as they are employed differently in some of the studies. A variety of programs are used as benchmarks, and C and Java dominate the benchmarks that include synthetic and real faults, as well as, single and multiple faults.

Some research opportunities were identified, such as to use many-objective optimisation algorithms, to analyse the quality of the sources of fault data and how such quality can impact on the effectiveness of SBFL methods, to use new sources of fault data, to address non-functional aspects of SBFL approaches as well as how such investigations can be evaluated, to analyse benchmarks with respect how fit they are to classes of faults, programming language, and fault nature (synthetic and real). We hope this study may improve motivation for new investigations and will support the decisions of the research field, serving as a reference and guidance for future SPFL methods.

References

[1] ISO/IEC/IEEE, Software and Systems Engineering, Software Testing, Part 1: Concepts and Definitions, International Standard 29119-1 (2013) 1–64.

[2] I. C. Society, P. Bourque, R. E. Fairley, Guide to the Software Engineering Body of Knowledge SWEBOK Version 3.0, IEEE Computer Society Press, Los Alamitos, CA, USA, 2014.

[3] I. Vessey, Expertise in debugging computer programs: An analysis of the content of verbal protocols, IEEE Trans. Syst. Man Cybern. 16 (5) (1986) 621–637.

[4] W. E. Wong, R. Gao, Y. Li, R. Abreu, F. Wotawa, A survey on software fault localization, IEEE Transactions on Software Engineering 42 (8) (2016) 707–740.

[5] P. S. Kochhar, X. Xia, D. Lo, S. Li, Practitioners' expectations on automated fault localization, in: Proceedings of the 25th International Symposium on

Software Testing and Analysis, ISSTA 2016, ACM, New York, NY, USA, 2016, pp. 165–176.

[6] B. Kitchenham, S. Charters, Guidelines for performing systematic literature reviews in software engineering (2007).

[7] ISO/IEC, Systems and Software Engineering, Systems and Software Assurance, Part 1: Concepts and Vocabulary, International Standard 15026-1.

[8] IEEE, Ieee standard classification for software anomalies - redline, IEEE Std 1044-2009 (Revision of IEEE Std 1044-1993) - Redline (2010) 1–25.

[9] T. Reps, T. Ball, M. Das, J. Larus, The use of program profiling for software maintenance with applications to the year 2000 problem, SIGSOFT Softw. Eng. Notes 22 (6) (1997) 432–449.

[10] M. J. Harrold, G. Rothermel, R. Wu, L. Yi, An empirical investigation of program spectra, SIGPLAN Not. 33 (7) (1998) 83–90.

[11] R. Abreu, P. Zoeteweij, A. J. C. van Gemund, On the accuracy of spectrum-based fault localization, in: Proceedings of the Testing: Academic and Industrial Conference Practice and Research Techniques - MUTATION, TAICPART-MUTATION '07, IEEE Computer Society, Washington, DC, USA, 2007, pp. 89–98.

[12] J. A. Jones, M. J. Harrold, J. Stasko, Visualization of test information to assist fault localization, in: Proceedings of the 24th International Conference on Software Engineering. ICSE 2002, 2002, pp. 467–477.

[13] R. Abreu, P. Zoeteweij, A. J. C. v. Gemund, An evaluation of similarity coefficients for software fault localization, in: Proceedings of the 12th Pacific Rim International Symposium on Dependable Computing, PRDC '06, IEEE Computer Society, Washington, DC, USA, 2006, pp. 39–46.

[14] L. Naish, H. J. Lee, K. Ramamohanarao, A model for spectra-based software diagnosis, ACM Trans. Softw. Eng. Methodol. 20 (3) (2011) 11:1–11:32.

[15] M. Harman, P. McMinn, J. T. de Souza, S. Yoo, Empirical software engineering and verification, in: B. Meyer, M. Nordio (Eds.), Empirical Software Engineering and Verification, Springer-Verlag, Berlin, Heidelberg, 2012, Ch. Search Based Software Engineering: Techniques, Taxonomy, Tutorial, pp. 1–59.

[16] M. Harman, B. F. Jones, Search-based software engineering, Information and Software Technology 43 (14) (2001) 833 – 839.

[17] P. Brereton, B. A. Kitchenham, D. Budgen, M. Turner, M. Khalil, Lessons from applying the systematic literature review process within the software engineering domain, J. Syst. Softw. 80 (4) (2007) 571–583.

[18] A. Zakari, S. P. Lee, K. A. Alam, R. Ahmad, Software fault localisation: a systematic mapping study, IET Software 13 (1) (2019) 60–74.

[19] K. Petersen, R. Feldt, S. Mujtaba, M. Mattsson, Systematic mapping studies in software engineering, in: Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering, EASE'08, British Computer Society, Swinton, UK, UK, 2008, pp. 68–77.

[20] K. Petersen, S. Vakkalanka, L. Kuzniarz, Guidelines for conducting systematic mapping studies in software engineering: An update, Information and Software Technology 64 (2015) 1 – 18.

[21] P. Agarwal, A. P. Agrawal, Fault-localization techniques for software systems: A literature review, SIGSOFT Softw. Eng. Notes 39 (5) (2014) 1–8.

[22] C. Wohlin, P. Runeson, P. A. da Mota Silveira Neto, E. Engström, I. do Carmo Machado, E. S. de Almeida, On the reliability of mapping studies in software engineering, Journal of Systems and Software 86 (10) (2013) 2594 – 2610.

[23] M. Harman, The current state and future of search based software engineering, in: 2007 Future of Software Engineering, FOSE '07, IEEE Computer Society, Washington, DC, USA, 2007, pp. 342–357.

[24] S. Ali, L. C. Briand, H. Hemmati, R. K. Panesar-Walawege, A systematic review of the application and empirical investigation of search-based test case generation, IEEE Trans. Softw. Eng. 36 (6) (2010) 742–762.

[25] A. Azarian, A. Siadat, P. Martin, A new strategy for automotive off-board diagnosis based on a meta-heuristic engine, Engineering Applications of Artificial Intelligence 24 (5) (2011) 733–747.

[26] W. Wang, P. You, W. Zhong, T. Xie, J. Xu, L. Zhong, X. Xiao, Optimization of guided wave sensors distribution along thin-walled small-diameter pipe, Zhongnan Daxue Xuebao (Ziran Kexue Ban)/Journal of Central South University (Science and Technology) 47 (7) (2016) 2254–2259.

[27] M. Harman, S. A. Mansouri, Y. Zhang, Search-based software engineering: Trends, techniques and applications, ACM Comput. Surv. 45 (1) (2012) 11:1–11:61.

[28] S. Biaz, Yiming Ji, Precise distributed localization algorithms for wireless networks, in: Sixth IEEE International Symposium on a World of Wireless Mobile and Multimedia Networks, 2005, pp. 388–394.

[29] B. Baudry, F. Fleurey, Y. Le Traon, Improving test suites for efficient fault localization, in: Proceedings of the 28th International Conference on Software Engineering, ICSE '06, Association for Computing Machinery, New York, NY, USA, 2006, p. 82–91.

[30] B. Liu, Lucia, S. Nejati, L. C. Briand, Improving fault localization for simulink models using search-based testing and prediction models, in: 2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER), 2017, pp. 359–370.

[31] F. Steimann, M. Frenkel, Improving coverage-based localization of multiple faults using algorithms from integer linear programming, in: 2012 IEEE 23rd International Symposium on Software Reliability Engineering, 2012, pp. 121–130.

[32] Y. Zhang, D. Lo, X. Xia, T. B. Le, G. Scanniello, J. Sun, Inferring links between concerns and methods with multi-abstraction vector space model, in: 2016 IEEE International Conference on Software Maintenance and Evolution (ICSME), 2016, pp. 110–121.

[33] L. Arcega, J. Font, C. Cetina, Evolutionary algorithm for bug localization in the reconfigurations of models at runtime, in: Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS '18, 2018, p. 90–100.

[34] B. Giroux, M. Chouteau, L. Laverdure, Évaluation du facteur de qualité sismique au barrage de carillon (québec), Revue Canadienne de Génie Civil 28 (2011) 496–508.

[35] C. Wohlin, Guidelines for snowballing in systematic literature studies and a replication in software engineering, in: Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering, EASE'14, ACM, New York, NY, USA, 2014, pp. 38:1–38:10.

[36] S. Pearson, J. Campos, R. Just, G. Fraser, R. Abreu, M. D. Ernst, D. Pang, B. Keller, Evaluating and improving fault localization, in: Proceedings of the 39th International Conference on Software Engineering, ICSE '17, IEEE Press, Piscataway, NJ, USA, 2017, pp. 609–620.

[37] J. W. Creswell, Research Design: Qualitative, Quantitative, and Mixed Methods Approaches, 8th Edition, SAGE Publications, 2018.

[38] C. Parnin, A. Orso, Are automated debugging techniques actually helping programmers?, in: Proceedings of the 2011 International Symposium on Software Testing and Analysis, ISSTA '11, ACM, New York, NY, USA, 2011, pp. 199–209.

[39] X. Li, W. E. Wong, R. Gao, L. Hu, S. Hosono, Genetic algorithm-based test generation for software product line with the integration of fault localization techniques, Empirical Software Engineering 23 (1) (2018) 1–51.

[40] S. Lal, A. Sureka, A static technique for fault localization using character n-gram based information retrieval model, in: Proceedings of the 5th India Software Engineering Conference, ISEC '12, ACM, New York, NY, USA, 2012, pp. 109–118.

[41] M. Hutchins, H. Foster, T. Goradia, T. Ostrand, Experiments of the effectiveness of dataflow- and controlflow-based test adequacy criteria, in: Proceedings of the 16th International Conference on Software Engineering, ICSE '94, IEEE Computer Society Press, Los Alamitos, CA, USA, 1994, pp. 191–200.

[42] H. Do, S. Elbaum, G. Rothermel, Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact, Empirical Softw. Engg. 10 (4) (2005) 405–435.

[43] R. Just, D. Jalali, M. D. Ernst, Defects4j: A database of existing faults to enable controlled testing studies for java programs, in: Proceedings of the 2014 International Symposium on Software Testing and Analysis, ISSTA 2014, ACM, New York, NY, USA, 2014, pp. 437–440.

[44] F. I. Vokolos, P. G. Frankl, Empirical evaluation of the textual differencing regression testing technique, in: Proceedings of the International Conference on Software Maintenance, ICSM '98, IEEE Computer Society, Washington, DC, USA, 1998, pp. 44–.

[45] S. H. Tan, J. Yi, Yulis, S. Mechtaev, A. Roychoudhury, Codeflaws: A programming competition benchmark for evaluating automated program repair tools, in: Proceedings of the 39th International Conference on Software Engineering Companion, ICSE-C '17, IEEE Press, Piscataway, NJ, USA, 2017, pp. 180–182.

[46] D. White, J. Mcdermott, M. Castelli, L. Manzoni, B. W. Goldman, G. Kronberger, W. Jaśkowski, U.-M. O'Reilly, S. Luke, Better gp benchmarks: Community survey results and proposals, Genetic Programming and Evolvable Machines 14 (2013) 3–29.

[47] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, A. Wesslén, Experimentation in Software Engineering: An Introduction, Kluwer Academic Publishers, Norwell, MA, USA, 2000.

Relevant Primary Studies

[R1] S. Wang, D. Lo, L. Jiang, Lucia, H. C. Lau, Search-based fault localization, in: Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering, ASE '11, IEEE Computer Society, Washington, DC, USA, 2011, pp. 556–559.

[R2] S. Yoo, Evolving human competitive spectra-based fault localisation techniques, in: Proceedings of the 4th Symposium on Search Based Software Engineering, SSBSE'12, Springer-Verlag, Berlin, Heidelberg, 2012, pp. 244–258.

[R3] L. Naish, Neelofar, K. Ramamohanarao, Multiple bug spectral fault localization using genetic programming, in: 2015 24th Australasian Software Engineering Conference, 2015, pp. 11–17.

[R4] D. Kang, J. Sohn, S. Yoo, Empirical evaluation of conditional operators in gp based fault localization, in: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '17, ACM, New York, NY, USA, 2017, pp. 1295–1302.

[R5] J. Sohn, S. Yoo, Fluccs: Using code and change metrics to improve fault localization, in: Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2017, ACM, New York, NY, USA, 2017, pp. 273–283.

[R6] D. de Freitas, P. Leitao-Junior, C. Camilo-Junior, A. Dantas, R. Harrison, Genetic programming-based composition of fault localization heuristics, in: Proceedings of the Eighth Brazilian Workshop on Search based Software Engineering, WESB'17, 2017.

[R7] S. Yoo, X. Xie, F.-C. Kuo, T. Y. Chen, M. Harman, Human competitiveness of genetic programming in spectrum-based fault localisation: Theoretical and empirical analysis, ACM Transactions on Software Engineering and Methodology 26 (1) (2017) 4:1–4:30.

[R8] D. de Freitas, P. Leitao-Junior, C. Camilo-Junior, R. Harrison, Evolutionary composition of customised fault localisation heuristics, in: Proceedings of the European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, ESANN'18, 2018.

[R9] K. Choi, J. Sohn, S. Yoo, Learning fault localisation for both humans and machines using multi-objective gp, in: Proceedings of the 10th Symposium on Search-Based Software Engineering, SSBSE 2018, Montpellier, France, 2018, pp. 349–355.

[R10] J. Kim, M. G. Epitropakis, S. Yoo, Learning without peeking: Secure multi-party computation genetic programming, in: Proceedings of the 10th Symposium on Search-Based Software Engineering, SSBSE 2018, Montpellier, France, 2018, pp. 246–261.

[R11] Y. Zheng, Z. Wang, X. Fan, X. Chen, Z. Yang, Localizing multiple software faults based on evolution algorithm, Journal of System and Software 139 (C) (2018) 107–123.

[R12] D. M. De-Freitas, P. S. Leitao-Junior, C. G. Camilo-Junior, R. Harrison, Mutation-based evolutionary fault localisation, in: 2018 IEEE Congress on Evolutionary Computation (CEC), 2018, pp. 1–8.

[R13] N. Neelofar, L. Naish, K. Ramamohanarao, Spectral-based fault localization using hyperbolic function, Software: Practice and Experience 48 (2018) 641–664.

[R14] J. Sohn, S. Yoo, Empirical evaluation of fault localisation using code and change metrics, IEEE Transactions on Software Engineering (2019) 1–1.