Investigating Fault Localization Techniques from Other Disciplines for Software Engineering

Árpád Beszédes
Department of Software Engineering, University of Szeged
Arpád tér 2., H-6720 Szeged, Hungary
beszedes@inf.u-szeged.hu

Keywords: Faults, Defects, Fault Localization, Software Fault Localization, Literature Review, Interdisciplinary Aspects.

Abstract: In many different engineering fields, fault localization means narrowing down the cause of a failure to a small number of suspicious components of the system. This activity is an important concern in many areas, and there have been a large number of techniques proposed to aid this activity. Some of the basic ideas used are common to different fields, but generally quite diverse approaches are applied. Our long-term goal with the presented research is to identify potential techniques from non-software domains that have not yet been fully leveraged to software faults, and investigate their applicability and adaptation to our field. We performed an analysis of related literature, not limiting the search to any specific engineering field, with the aim to find solutions in non-software areas that could be most successfully adapted to software fault localization. We found out that few areas have significant literature in the topic that are good candidates for adaptation (computer networks, for instance), and that although some classes of methods are less suitable, there are useful ideas in almost all fields that could potentially be reused. As an example of potential novel techniques for software fault localization, we present three concrete techniques from other fields and how they could potentially be adapted.

1 INTRODUCTION

Failures in complex systems can cause damage to the environment, people’s health and lives, or the operation of businesses and governments. Hence, possible underlying faults in the respective engineering disciplines are a high priority concern.

These complex systems may be mechanical, electrical, software-driven, or any combination thereof, but there is one common subtopic, the central theme of this article, fault localization. In a very general sense, fault localization in complex (software and non-software) systems means identifying components (parts, modules, software code parts, etc.) of the system that are responsible for a specific observed failure or set of failures. Note, that fault localization is closely related to the notion of fault detection, the main difference being that the latter merely shows that there is a fault somewhere in the system but does not tell where.¹

Due to the importance of fault localization, a number of different discipline- and domain-specific solutions have been proposed to aid this process in an automatic or semi-automatic manner. In some situations, fault detection and localization are not separately handled, but our goal with the present work was to concentrate on the fault localization aspects only.

In this work, our main concern are software faults. In particular, our aim is to find out whether there are techniques used in non-software domains that have not yet been considered for software faults but could be adapted to the same. We deal with (semi)automatic fault localization techniques from various engineering disciplines, and start with an interdisciplinary analysis of the topic. To this end, we performed a systematic literature review with a very specific goal, to discover their applicability to software fault localization. We then elaborate on the possible enhancements to existing techniques or devising novel approaches that were successful in other disciplines but has not yet been considered for software.

In any of the mentioned engineering areas, systems tend to be large and complex, and they are often connected to each other, forming even more complex systems-of-systems. Consequently, it may be very difficult to localize the origin (root cause) of occurring failures. Hence, various fields have developed

¹In related literature, there is often a confusion about these terms and their exact meaning might be different across various disciplines.
approaches to automate the fault localization process. Naturally, each field deals with its peculiarities and many of the techniques are domain-dependent, yet we found out that there are some similarities across disciplines. Furthermore, some of the methods are generic and could be applied, theoretically, to any engineering field and fault localization problem.

Software fault localization has a large literature covering many different subtopics; see (Wong et al., 2016; Parmar and Patel, 2016). However, other areas, for instance, aerospace or electronics, have much longer histories and hence might provide ideas for advancing the software field. Next, a lot of research has been performed to design effective software fault localization algorithms and propose their use in different phases of the software process, most notably debugging. But, related research suggests that the practical applicability of research results in this area is still limited (Kochhar et al., 2016), and further research is needed to achieve more widespread use of automatic software fault localization by practitioners. We also noticed that recent results in this area are able to only marginally improve the effectiveness of previous techniques because it is increasingly difficult to make significant and highly novel contributions.

Hence the motivation for the present work; to investigate other engineering fields and find out if they employ techniques that could be adapted to software and thus advance this field. We found that in some cases there are barriers to the adoption of such techniques due to the fundamental differences in these (software and non-software) systems. Some of them arise from the fact that software systems are much more intangible, but also there are notable differences in how they are described and handled, e.g., types of behavioral models used to describe the expected behavior of the systems. In other cases, however, the techniques or some underlying ideas could be successfully adapted to software.

In previous work (Beszédes, 2019), we presented details of an interdisciplinary survey of fault localization techniques to aid software engineering. After providing the necessary background information and detailed assessment criteria, we presented the categorization of the identified methods according to several major fields. Also, a set of possible relationships between the areas has been presented. In the present work, we introduce three concrete examples of how techniques successfully used in other fields could potentially be adapted to software faults. Our aim with this approach is to illustrate the relevance of this research, but further details and more investigation are needed to elaborate on the actual implementability of the methods.

The paper is organized as follows. Section 2 describes the approach we used for the literature survey. In Section 3, we present the results of the literature analysis with our initial evaluation and example adoptable techniques. Finally, Section 4 concludes.

2 LITERATURE SURVEY

In this section, we briefly overview the process we followed in reviewing fault localization literature in various engineering fields.

In the first phase, we identified the most relevant papers with the help of the google, google scholar, ResearchGate, Mendeley and Scopus systems. We used a combination of generic search terms like “fault localization” and specific keywords that we expected to be relevant for our search: networks, electronics, engineering, operations, systems, etc. We also applied different variations and synonyms to these terms, which included localizing faults, failure diagnosis, problem diagnosis, error localization, and similar. This way, we were able to identify related works in many different engineering fields.

We restricted the search results to full-text scientific publications. We aimed at limiting the results to publications that appeared in peer-reviewed journals or conferences, however there were few exceptions such as doctoral theses and technical reports. The next filtering we applied was to limit the list to papers that correspond to some of the following categories: software-related, generic algorithms, and methods from loosely related engineering fields. For example, pure mathematical methods and approaches in non-related scientific branches like biomedicine, navigation, linguistics or other, were removed.

In addition to the repository and proceeding searches, we also performed lightweight snowballing (Wohlin, 2014) to mitigate the risk of omitting relevant literature. Finally, we consolidated the results by organizing the works by specific research groups or authors and concentrating on a few relevant reports by the same team.

We then performed an initial classification of the papers based on what fundamental area they belong to. We used a simple classification in this respect: software, networking, other engineering and various/generic. There exists a large amount of publications that deal with fault localization in computer networks, hence we established a separate category for this area. The other engineering category includes all methods that belong to a specific engineering field other than software or networks. Finally, there are some approaches that are not limited to any specific
field (although some of them include one or more example applications); in this sense, they are generic. We used the same category to denote methods belonging to some other various fields.

Finally, we extended the classification using other criteria which might be additionally useful to determine the methods’ usability in the software domain. These criteria included:

- The basic approach on which the method is based on: machine learning, statistical analysis, entropy-based, etc.
- Single or multiple faults are handled.
- Basic type of data the method relies on.
- The need for a behavior model.
- Other properties of the related studies such as kind of empirical measurements, size of data sets, and availability of data or implementation.

Details about these criteria and the analysis outcomes are omitted from this paper due to space considerations, but all the data are available in previous work (Beszédes, 2019).

3 ANALYSIS AND PROPOSED NEW TECHNIQUES

Our initial findings indicate that it is not easy to pinpoint only a few candidate methods, rather many of them may provide interesting ideas, even if not the complete method is adapted.

Below, we provide an overview of the most notable techniques we identified and their initial evaluation for each of the main fundamental areas. Software is intentionally omitted from this list, as we wanted to find approaches not yet existing in our field. For each of the three relevant categories (Networking, Other Engineering and Generic Methods), we propose specific new software fault localization methods based on the related approaches. These can serve as examples of interdisciplinary application of a technique, but this does not necessarily mean that we cannot benefit from other, less directly related methods.

3.1 Networking

The main representative literature we identified in the networking category are: (Steinder and Sethi, 2004c), (Aleksan and Sayenko, 2014), (Yu et al., 2010), (Brodie et al., 2002), (Brodie et al., 2003), (Chao et al., 1999), (Chen et al., 2004), (Cheng et al., 2010), (Deng et al., 1993), (Fecko and Steinder, 2001), (Lu et al., 2013a), (Hood and Ji, 1997), (Kant et al., 2004), (Katzela and Schwartz, 1995), (Kompella et al., 2005), (Lu et al., 2013b), (Mohamed, 2017), (Natu and Sethi, 2005), (Natu and Sethi, 2006), (Natu and Sethi, 2007a), (Natu and Sethi, 2007b), (Rish et al., 2004), (Tang et al., 2005), (Traczzyk, 2004), (Wang et al., 2012), (Zhang et al., 2011), (Boubour et al., 1997), (Aghasaryan et al., 1997), (Aghasaryan et al., 1998), (Steinder and Sethi, 2004a), (Steinder and Sethi, 2004b), (Ben-Haim, 1980).

A common element of network fault localization is the use of probabilistic approaches (such as conditional probabilities and Bayes networks) (Chao et al., 1999; Tang et al., 2005; Steinder and Sethi, 2004b), and machine learning (Chen et al., 2004; Deng et al., 1993; Hood and Ji, 1997), which could probably be adapted to software faults.

The probing method in computer networks, e.g. (Steinder and Sethi, 2004c), is an almost direct analogy to software fault localization (a probe is a program that executes on a particular network node and sends commands or transactions to the other elements of the network, and the responses are observed and their various properties are measured). A network node may correspond to a software component, a probe can be seen as a test case, and the responses from the network can be identified as the dynamic behavior of the system by executing the test cases.

In this case, the Spectrum-Based Fault Localization class of methods (Abreu et al., 2009; Parmar and Patel, 2016; Wong et al., 2016) could be naturally extended using results from networking. Here, the basis of the algorithms is the so-called program spectrum, which in its basic form is essentially a binary matrix consisting of the test cases in its rows and the program elements in the columns. A 1 in a matrix element represents that the test case executes the corresponding code element. Then, based on the test case outcomes, various statistics are computed for each program element about the number of failing and passing test case executions traversing it, and the most suspicious code elements are reported to the user (many failing test cases and few passing ones make a code element more suspicious).

Approaches by Yu et al. (Yu et al., 2010), Brodie et al. (Brodie et al., 2002; Brodie et al., 2003), Cheng et al. (Cheng et al., 2010), Natu et al. (Natu and Sethi, 2005; Natu and Sethi, 2006; Natu and Sethi, 2007a; Natu and Sethi, 2007b) provide various optimizations to the basic probing approaches, which are good candidates for adaptation to (spectrum-based) software fault localization.

In particular, the following can be an optimized approach to fault localization based on the results
from publications in the networking field. These steps could be for optimal test suite construction (here, a test case corresponds to a probe in networking):

1. Start from an initial test case set, e.g., optimizing for high code coverage.
2. Determining the diagnosis ability of this set. This can be based on some property using entropy or some other measurement. Similar results are already available in software fault localization (Perez et al., 2017; Perez and Abreu, 2018).
3. Finding the minimal set of test cases, based on some heuristics such as greedy search.

Based on the mentioned works, the program spectra (coverage matrix) can be extended to a Bayesian network that encodes probabilistic dependencies between the possible faults (causes) and the test outcomes (symptoms) (Brodie et al., 2002).

### 3.2 Other Engineering Fields

We identified the engineering fields of aerospace and (power) electronics that could potentially be the most useful for our purposes, along with a few additional techniques from specific disciplines like oil pipelines and chemistry. For these fields, our main findings are: aerospace (Adamovits and Pagurek, 1993; Balaban et al., 2007; W. Dries, 1990), power electronics (Benbouzid et al., 1999; Beschta et al., 1993; Poon, 2015; Tanwani et al., 2011), electronics (Poischl and Wotawa, 2006), oil pipelines (Digernes, 1980), and chemistry (Venkatasubramanian et al., 1990).

The most promising novel approach to software fault localization is to use behavioral software models against which the actual behavior could be compared to. A behavioral model essentially describes the expected functionality of the system which might be expressed using various formalisms, either graphical or textual. This area is often referred to as Model-Based Diagnosis. This concept has not yet been investigated in depth for our field. The closest approach is Abreu et al.’s work on using Bayesian framework for software fault localization (Abreu et al., 2009b), however this technique does not incorporate other components of model-based diagnosis overviewed below.

In the aerospace industry, the use of model based reasoning, a branch of artificial intelligence, is frequent (Adamovits and Pagurek, 1993; Balaban et al., 2007; W. Dries, 1990). In power electronics (Poon, 2015; Benbouzid et al., 1999; Beschta et al., 1993), the methods also frequently utilize various models describing the system. Although somewhat different, but some approaches to fault localization in electronic circuits are using behavioral models combined with simulation. In addition, in some cases the description of the hardware is done in a similar way to software source code (Poischl and Wotawa, 2006), which makes it possible more easier to adapt to our field. Other areas we investigated often use simulation and probabilistic approaches as well (Digernes, 1980), or machine learning with neural networks (Venkatasubramanian et al., 1990), and in these cases a model of the system is required as well. Often, advanced concepts are applied in these areas such as Kalman filters to increase the accuracy of fault estimates.

A model-based approach to fault localization in software requires reliable behavioral models that describe the expected behavior. This is then compared to the observed data from system under examination. Models are in some cases already available; especially, in the case of critical systems. Here, systematic and rigorous requirement elicitation, fault mode and effect analysis, and test design is done. Models and their languages depend on the domain and the modeling approach used. Although the reasoning components of the system may also include experimental knowledge, the model can largely be seen as a black box, which encodes the required behavior and is determined during early software engineering phases. For successful model-based diagnosis, models need to be accurate and independent of the reasoning structure. Completeness is also important, but the faulty behaviour is generally little-known. Models are often encoded in the form of a simulation.

Once the model is available, model-based approach to fault localization in software would follow the usual approach in model-based diagnosis: given observations of the system, the system is simulated using the model, and the observations actually made are compared to the observations predicted by the simulation. The following specific subtasks would be required in general: data acquisition (producing dynamic operational data from program executions, which can be performed using existing approaches such as profiling, logging and instrumentation), fault detection (performing the tests and comparison to the model), hypothesis generation and pruning (producing and analyzing “infection chains” (Zeller, 2005) from defects to the symptoms), and hypothesis validation (through formal model analysis or simulation).

### 3.3 Generic Methods

In the generic category, we identified the following main techniques: (Isermann, 1984), (Massoumia et al., 1986), (Svárd, 2012), (Varga, 2003), (Lerner and Parr, 2000), (Olivier-Maget et al., 2009), (Shchekotykhin et al., 2016), (Mehra and Peschon, 2015; Benbouzid et al., 1999; Beschta et al., 1993), probabilistic approaches as well (Digernes, 1980), or machine learning with neural networks (Venkatasubramanian et al., 1990), and in these cases a model of the system is required as well. Often, advanced concepts are applied in these areas such as Kalman filters to increase the accuracy of fault estimates.

A model-based approach to fault localization in software requires reliable behavioral models that describe the expected behavior. This is then compared to the observed data from system under examination. Models are in some cases already available; especially, in the case of critical systems. Here, systematic and rigorous requirement elicitation, fault mode and effect analysis, and test design is done. Models and their languages depend on the domain and the modeling approach used. Although the reasoning components of the system may also include experimental knowledge, the model can largely be seen as a black box, which encodes the required behavior and is determined during early software engineering phases. For successful model-based diagnosis, models need to be accurate and independent of the reasoning structure. Completeness is also important, but the faulty behaviour is generally little-known. Models are often encoded in the form of a simulation.

Once the model is available, model-based approach to fault localization in software would follow the usual approach in model-based diagnosis: given observations of the system, the system is simulated using the model, and the observations actually made are compared to the observations predicted by the simulation. The following specific subtasks would be required in general: data acquisition (producing dynamic operational data from program executions, which can be performed using existing approaches such as profiling, logging and instrumentation), fault detection (performing the tests and comparison to the model), hypothesis generation and pruning (producing and analyzing “infection chains” (Zeller, 2005) from defects to the symptoms), and hypothesis validation (through formal model analysis or simulation).
It is worthwhile to note that some techniques we listed here (e.g. Kleer et al. (de Kleer, 2009; de Kleer and Williams, 1987)) have their main application in the mentioned fields from previous sections such as electronic circuits and computer networks, and are based on techniques such as entropy minimization and probabilistic approach. In particular, Bayesian framework is the basis for Kleer et al.'s technique, which has also been reused for software (Abreu et al., 2009b).

Model-Based Diagnosis, mentioned in the previous section, aims at finding the fault of an observed system based on the knowledge about the system’s expected behavior, which can be discussed in a generic, domain-independent manner (Shchekotykhin et al., 2016; Frank, 1996; de Kleer, 2009; de Kleer and Williams, 1987). Many of these model the system as a process, and hence process analysis approaches are used from control theory (Isermann, 1984; Massoumnia et al., 1986; Svärd, 2012; Venkatasubramanian et al., 1990; Mehra and Peschon, 1971).

For software fault localization, we propose to further investigate the combination of model based approaches with data driven methods (which process a large amount of data from the systems output and are based on training data for a correctly working system). Tidridi et al. (Tidriri et al., 2016) present an overview of the approach. We believe this could be a promising way because a reliable and complete model is often not available for software, but the operational data from software executions is easily obtainable, e.g. from logs. Tidridi et al. also refer other related works in this area, which can be useful sources for more information about this set of techniques.

In particular, in a data driven approach, a large amount of operational data (from logs, etc) are collected in which the fault is first detected (whether the system behavior matches the expected one) and then classified (determining the type of the fault). Typical classification types are supervised approaches based on manually prepared training data (e.g., using Neural Networks), and unsupervised methods such as various statistical techniques like Principal Component Analysis. These techniques can help model-based diagnosis is various ways, because in practice it is very difficult to develop accurate mathematical models for large and complex software systems. For example, the primary form of behavior differences to be observed by a model-based approach (see the previous section) can be generated and learned from a large amount of dynamic operational data.

4 CONCLUSIONS

The purpose of the presented work was to demonstrate that there is a lot of potential in interdisciplinary application of fault localization techniques from several different engineering fields to software faults. We performed a literature review involving various fields, computer networks, aerospace, (power) electronics, etc., and found that although in many cases the too large differences in the domains permit the techniques’ application to our field, there are several ideas that could be leveraged for software.

Preliminary results presented in preceding sections could provide a starting point for additional analysis of the techniques, and the selection of the most promising approaches for further consideration. We also present three concrete approaches and how they could be adapted to software fault localization.

Although we performed the literature analysis in a systematic way, we cannot claim its completeness. Indeed, we did not dive into the details of any particular subfield or set of techniques. Instead, we concentrated on locating the most important approaches from various fields. Nevertheless, we believe that the survey in its present state is suitable for us to start designing in detail novel approaches to software fault localization, and for other readers to obtain a wider view of this important and diverse topic.

As future work, we will continue searching for other related techniques, extend the assessment with additional aspects and evaluate the most promising approaches in more detail. We will eventually start implementing the new techniques for software fault localization and compare their performance to other, already established techniques in this field.

Detailed assessment data of the literature survey are available from the author.

ACKNOWLEDGEMENTS

Árpád Beszédes was supported by the János Bolyai Research Scholarship of the Hungarian Academy of Sciences. The author would like to thank the supporting work of Öttö Eötvös and Brúno Ilovai who were supported by project EFOP-3.6.3-VEKOP-16-2017-0002, co-funded by the European Social Fund. Ministry of Human Capacities, Hungary grant 20391-3/2018/FEKUSTRAT is acknowledged.
REFERENCES


