Do Automated Fault Localization Techniques Work For REST APIs?

Aakash Kulkarni
School of EECS
Oregon State University
Corvallis, United States
kulkaraa@oregonstate.edu

Soon Song Cheok
School of EECS
Oregon State University
Corvallis, United States
cheoks@oregonstate.edu

Shreyes Joshi
School of EECS
Oregon State University
Corvallis, United States
joshish@oregonstate.edu

***Abstract*—**
***Index Terms*—**

I. Introduction

REST APIs (Representational State Transfer) have become backbone of the modern web and cloud applications. They facilitate seamless interactions between client and server through stateless communication, enabling services to be scalable, reliable, and easily integrateable [1] [2]. Basically, REST APIs are a set of rules and standards defined by OpenAPI [3] used to enable communication between different software applications over the internet. They are built around the use of standard HTTP methods such as GET, POST, PUT, and DELETE to interact with resources, which are any kind of data or service that can be named on a network. Given their critical role, the effective identification and resolution of faults within REST APIs remain a significant challenge [4], prompting the need for research on how fault localization techniques work to their unique structure and functionality.

As systems that rely on REST APIs grow in scale and complexity [5] , minor faults can escalate into major disruptions, impacting user experience and business operations. This research is motivated by the need to evaluate how well current fault localization techniques perform in the unique context of REST APIs. The goal is to determine if these techniques can indeed be applied effectively to REST APIs and, if so, to explore which types of faults are more amenable to being localized. This understanding could potentially allow developers to more efficiently diagnose and address issues, thereby enhancing the stability and performance of REST API-based systems.

Fault localization is a well-established area of research within software engineering, traditionally concentrated on more conventional software systems. REST APIs, however, present distinct challenges that complicate fault localization due to their composition and operational dynamics. Additionally, the architecture of REST APIs often involves diverse artifacts that are not source code, such as configuration files, API specifications, and database schemas.

The primary objective of this research is to evaluate the effectiveness of existing fault localization techniques within the unique context of REST APIs, which feature a mix of code and non-code artifacts. This study will systematically apply established fault localization methods like BLUES [6] and LLMAO [7] to a dataset of REST API faults. This dataset will be meticulously developed to represent a wide range of faults typical in REST APIs and classified according to an existing comprehensive taxonomy derived from previous research.

This approach will enable us to assess the applicability of these fault localization techniques to REST APIs by determining how effectively they can identify and localize different types of faults. The analysis will provide insights into whether traditional fault localization methods are suited to the complexities of REST APIs, especially considering their unique structural and functional characteristics.

Our assumptions:

- *Assumption:* Assume that the selected fault localization techniques are suitable for application to REST APIs, despite their original design for conventional software systems. This includes the assumption that these techniques can handle the unique challenges posed by REST APIs, such as dealing with non-code artifacts.

The primary research questions are formulated as follows:

- **RQ1:** What categories of faults are most effectively localized by current techniques?
- **RQ2:** Which fault localization technique is highly effective in the context of REST APIs?

For the evaluation dataset, we will manually select the issues from the GitHub services and their fixes from REST API projects that utilize Spring Boot or Jersey frameworks. These GitHub services are used in the paper *Generating REST API Specifications through Static Analysis* by Manish et al [8] This dataset will encompass various categories of faults identified in REST APIs. By analyzing repositories and commit histories, we will extract specific instances where bugs have been documented and subsequently fixed. The categorization of these faults will be categorized with an established taxonomy of API faults [9] , ensuring that each bug is classified according to its nature and impact. This rigorous dataset compilation aims to provide a comprehensive basis for testing fault localization techniques within a controlled, relevant environment.

For evaluation of fault localization techniques using the dataset derived from REST APIs employing Spring Boot and Jersey frameworks will be measured by Top-k accuracy, and EXAM score. These metrics will collectively provide a thorough assessment of each fault localization technique's ability to detect and accurately categorize faults in REST APIs, contributing to a detailed understanding of their practical utility and areas for improvement.

II. Background and Motivation

A. *Importance of REST APIs*

REST APIs (Representational State Transfer APIs) are central to modern web and cloud applications, serving as critical conduits for data exchange and system integration. They leverage standard HTTP methods like GET, POST, PUT, and DELETE to interact with networked resources, making them integral to the architecture of distributed systems. The scalability, reliability, and ease of integration provided by REST APIs facilitate seamless interactions between clients and servers, thereby enhancing the performance and flexibility of complex software ecosystems. Their widespread adoption underscores their importance in today's technology landscape, where rapid communication and data accessibility are paramount.

B. *Challenges of Fault Localization in REST APIs*

While REST APIs have simplified the development and management of modern applications, they introduce specific challenges that complicate fault localization. The architecture often involves non-code artifacts such as API specifications, configuration files, and database schemas, which are not traditionally addresses by fault localization techniques developed for code-centric applications.

C. *Gaps in Current Research*

Fault localization is a well-established research area within software engineering, focusing primarily on identifying the locations of faults in conventional software systems to reduce debugging time and enhance system reliability. However, the unique operational dynamics and architectural complexities of REST APIs pose new challenges that are not fully addressed by existing fault localization techniques. Previous studies have developed fault taxonomies for REST APIs which are helpful if the FL techniques can localize faults to understand what categories of faults can be localized. There remains a significant knowledge gap regarding the effectiveness of these techniques in accurately identifying and localizing faults in REST APIs, particularly those involving non-code artifacts.

D. *Need for Targeted Research on Fault Localization in REST APIs*

This research is motivated by the critical need to evaluate and understand the performance of existing fault localization techniques within the REST API context. By determining how effectively these techniques can identify and localize faults in REST APIs, developers can gain valuable insights that could lead to quicker and more accurate fault diagnosis. This is particularly important as even minor faults can escalate into major disruptions in REST API-dependent systems, affecting user experience and operational efficiency. Addressing this gap will not only contribute to the field by enhancing the robustness and reliability of REST APIs but also by supporting the development of more sophisticated tools and methodologies tailored to the needs of modern software architectures.

E. *Research Objectives*

The primary objective of this research is to systematically assess the applicability of established fault localization techniques—namely BLUES and LLMAO to a curated dataset of REST API faults. This dataset will be developed to encompass a broad spectrum of typical faults in REST APIs and will be classified according to an existing comprehensive taxonomy. The insights gained from this study are expected to reveal whether traditional fault localization methods are suitable for the complex environments of REST APIs and may guide future efforts in refining these techniques or developing new approaches specifically designed for REST API ecosystems.

III. Related Work

Fault localization techniques have long been an integral part of software maintenance, aiding developers by pinpointing the origins of failures within software systems. A variety of approaches have been explored [10], including Information Retrieval (IR) based methods, Spectrum-Based Fault Localization (SFL), and more comprehensive software engineering tools.

Information Retrieval Based Fault Localization: The work by Klaus Changsun Youm et al introduces an advanced IR-based fault localization method called BLIA, which integrates bug reports and code change history to enhance the localization process [11] [12]. BLIA notably improves the accuracy of locating bugs by analyzing texts, stack traces, and comments within bug reports alongside source file structures and version histories. This method has shown significant improvements over previous tools such as BugLocator and BLUiR, particularly at the file and method levels, by refining the granularity of localization results. This approach underscores the potential of IR-based methods in handling complex software systems, aligning with the need to explore such techniques within REST API environments.

The Jaguar tool, described by Ribeiro et al, utilizes SBFL technique to identify faulty code excerpts in Java programs [13]. By employing both control-flow and data-flow spectra and integrating the lightweight ba-dua coverage tool, Jaguar efficiently supports the analysis of large programs. The capability of Jaguar to provide detailed visualizations of suspicious program elements highlights the effectiveness of SFL in traditional software environments. The distinction between data-flow and control-flow spectra offers a nuanced view of fault localization that could be pertinent to analyzing REST APIs, known for their complex interactions and stateless operations.

Debugging in IDEs, particularly using tools like Visual Studio Code for web applications, illustrates the practical aspects of fault localization in real-world scenarios [14]. The capabilities of IDEs to facilitate debugging through breakpoints and variable inspections are directly relevant to the challenges faced in REST API development, where faults may not only arise from code but also from API configurations and interactions.

IV. Approach

A. Dataset Creation and Preparation

To effectively assess the applicability of fault localization techniques to REST APIs, In our research, we utilize a set of REST API services previously employed in the study ”Generating REST API Specifications through Static Analysis” by Manish et al [8]. These services are chosen due to their diverse characteristics and well-documented configurations, making them suitable subjects for evaluating fault localization techniques. The faults from these services will be categorized based on a previously established taxonomy of REST API faults, ensuring that each fault type is adequately represented. We are three graduate students with knowledge on REST API, we have selected the issues on GitHub Repositories, and we have the read the messages of their issues and saw what were the changes made and given it a fault type. We all have gone through all the services and then we together consolidated into one single list where we found 26 Faults which we can test using the Fault Localization Techniques.

B. Selection of Fault Localization Techniques

The study will focus on two primary fault localization techniques that have shown promise in traditional software contexts but whose effectiveness in REST APIs remains underexplored:

1) **BLUES:** [6] BLUES is developed by a group of researchers Manish et al which takes Bug report as input and generates the suspicious ranked list of faulty code.
2) **LLMAO:** [7] LLMAO is developed by a group of researchers Aidan et al, where LLMAO is first LLM (Large Language Model) based for fault localization and which takes no inputs. But it finds the difference between two commits to see where the bug was introduced and creates a ranked list of faulty code.

Both the Fault Localization Techniques are known for their effective finding of the faulty code. And also both the techniques are new.

C. Integration with REST API Characteristics

Given the unique challenges posed by REST APIs, such as stateless operations and the presence of non-code artifacts (e.g., configuration files and API specifications), our approach will also consider these elements:

- **Non-Code Artifacts**: Special attention will be given to faults that may originate from or affect non-code artifacts.

TABLE I
List of Repositories

No.	Repository Name	# of Faults
1	management-api-for-apache-cassandra	2
2	cwa-verification-server	7
3	digdag	2
4	ohsome-api	10
5	kafka-rest	3

V. Evaluation

A. Dataset

To conduct this research, a dataset is required for evaluation. The dataset contains REST API faults, their sources, and other relevant information necessary like Bug Reports to evaluate FL techniques.

The dataset we started with had GitHub APIs from 15 different REST API repositories from the services were management-api-for-apache-cassandra, catwatch, cwa-verification-server, digdag, enviroCar-server, features-service, gravitee-api-management, kafka-rest, ocvn, ohsome-api, proxyprint-kitchen, quartz-manager, restcountries, senzing-api-server, and Ur-Codebin-API. The process of filtering out issues not marked as ’bugs’ and ’closed,’ leaving only the set of issues that have been fixed details I. Issues involving code changes are collected and then further filtered through manual analysis to select those applicable to this research.

This dataset provides a benchmark to evaluate FL techniques on REST API faults.

The fault taxonomy utilized in our study delineates four major fault types, each encompassing various sub-categories as outlined in [9]. Collectively, these categories provide a comprehensive framework, dividing the faults into a total of 19 distinct categories. This detailed classification aids in systematically analyzing and categorizing the faults encountered during the testing of REST APIs to create the Table II

TABLE II
Fault Data

No.	Fault Type	# of Faults
1	DB Operations/ Optimistic Lock Exception	1
2	DB Operations/ Unique Index or Primary Key	2
3	Incomplete Specification/Form/Format	1
4	Framework Misconfiguration/ Rejected Calls	2
5	Harcoded	1
6	Incomplete Specification/Custom Constraint/Semantic	1
7	Inconsistent Behavior/ Unexpected 500	1
8	Inconsistent Behavior/ Multiple Execution Fault	1
9	Incomplete Specification/Custom Constraint/Invalid Combination	5
10	Incomplete Specification/Form/Missing or Null values	3
11	Incomplete Specification/Form/Type	1
12	Unsupported Code	3

B. Metrics

There are several metrics that can be used to evaluate how effective FL techniques are at localizing REST API faults.

In this research, two primary metrics to consider are **Top-k accuracy**, and **EXAM**.

Top-k accuracy: This metric measures the percentage of faulty statements found within the top k positions of the ranking generated by an FL technique. The ranking positions are based on how likely each statement is considered faulty according to the fault localization algorithm. This metric is useful for evaluating how effective FL techniques are at localizing faults. In this research, several values of k are used to measure the FL technique's performance, such as top-1, top-3, top-5, and top-10.
This metric measures the percentage of times the actual faulty statements are within the top k positions of a ranked list produced by the fault localization technique. Mathematically, it is expressed as:

$$\text{Top-k} = \frac{\text{\#faults found in top } k \text{ positions}}{\text{Total number of faults}} \times 100\%$$

This metric evaluates the effectiveness of fault localization techniques by their ability to rank the actual faulty statements highly on the list. Commonly used values for k include 1, 3, 5, and 10.

EXAM score: This metric is defined as the percentage of the total statements a developer needs to examine before finding the fault location. A low EXAM score indicates that the fault localization technique was able to rank faulty statements highly, reducing the amount of code that needs to be inspected. This reflects an effective fault localization technique that quickly directs developers to the most likely faulty areas. However, the EXAM score does not account for multiple faults, limiting its ability to measure the effectiveness of an FL technique for multiple faults.

The EXAM score is calculated as the percentage of the code that must be examined until the first occurrence of a fault is encountered. It is defined as:

$$\text{EXAM Score} = \frac{\text{Position offault in the ranked list}}{\text{Total number of statements}} \times 100\%$$

A lower EXAM score indicates a more effective fault localization technique, as it suggests that fewer statements need to be inspected before finding the fault. This metric is particularly useful for assessing the efficiency of a technique in directing developers quickly to the likely fault locations.

Preicision: This metric measures the fraction of actually faulty statements out of the total reported faulty statements. It provides a measurement indicating how well an FL technique can localize faults. However, it doesn't capture an FL technique's capability to localize all faults.

The metrics are effective measures of FL technique performance. They will be used to measure how well FL techniques can localize REST API faults.

C. Experiment Procedure

The objective of our experiments is to systematically evaluate the effectiveness of fault localization techniques on REST APIs. Below, we outline the structured procedure for executing these experiments.

D. Setup

For each REST API A:

- Identify and enumerate all known faults F within the API.
- Select fault localization techniques T that are applicable to REST APIs. Each technique will be tested for its ability to accurately localize faults within the API.

The following algorithm details the steps involved in evaluating each fault localization technique, formatted to fit within a two-column document layout:

Fig. 1. Evaluate Fault Localization Techniques

```
for each API $A$ do
  for each Fault $F$ in $A$ do
    for each Technique $T$ that supports $A$ do
      Localize Fault:
      $R \leftarrow$ localize(Fault $F$, using $T$)
      $o/p \leftarrow$ compare($R$, GroundTruth)
      if $o/p$ is true then
        localize(Fault)
      else
        unlocalize(Fault)
      end if
    end for
  end for
end for
Record the number of Faults localized and not localized
```

Data Analysis

- Aggregate data to calculate overall performance metrics.
- Analyze data to identify conditions under which each technique performs best.

Reporting Results The results of the experiments will be presented in a comparative format, highlighting the effectiveness of each technique, based on:

1) Number of faults correctly localized versus not localized.
2) Statistical analysis of performance metrics across different APIs and fault types.
3) Discussion of technique-specific strengths or weaknesses observed during the experiments.

E. Results

VI. Discussion and Threats to Validity

VII. Contributions

References

[1] L. Li, W. Chou, W. Zhou, and M. Luo, "Design patterns and extensibility of rest api for networking applications," *IEEE Transactions on Network and Service Management*, vol. 13, pp. 154–167, 2016.

[2] A. Neumann, N. Laranjeiro, and J. Bernardino, "An analysis of public rest web service apis," *IEEE Transactions on Services Computing*, vol. 14, pp. 957–970, 2018.

[3] H. Ed-Douibi, J. Izquierdo, and J. Cabot, "Openapitouml: A tool to generate uml models from openapi definitions," in *International Conference on Web Engineering*. Cham, Switzerland: Springer, 2018, pp. 487–491.

[4] A. Barbir, C. Hobbs, E. Bertino, F. Hirsch, and L. Martino, "Challenges of testing web services and security in soa implementations," in *Test and Analysis of Web Services*. Berlin/Heidelberg, Germany: Springer, 2007, pp. 395–440.

[5] R. Khare and R. Taylor, "Extending the representational state transfer (rest) architectural style for decentralized systems," in *Proceedings of the 26th International Conference on Software Engineering*, 2004, pp. 428–437.

[6] M. Motwani and Y. Brun, "Better automatic program repair by using bug reports and tests together," in *Proceedings of the 45th International Conference on Software Engineering*, ser. ICSE '23. IEEE Press, 2023. [Online]. Available: https://doi.org/10.1109/ICSE48619.2023.00109

[7] A. Z. H. Yang, C. Le Goues, R. Martins, and V. Hellendoorn, "Large language models for test-free fault localization," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24. New York, NY, USA: Association for Computing Machinery, 2024. [Online]. Available: https://doi.org/10.1145/3597503.3623342

[8] R. Huang, M. Motwani, I. Martinez, and A. Orso, "Generating rest api specifications through static analysis," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24. New York, NY, USA: Association for Computing Machinery, 2024. [Online]. Available: https://doi.org/10.1145/3597503.3639137

[9] B. Marculescu, M. Zhang, and A. Arcuri, "On the faults found in rest apis by automated test generation," *ACM Transactions on Software Engineering and Methodology*, vol. 31, pp. 1–43, 07 2022.

[10] W. E. Wong, R. Gao, Y. Li, R. Abreu, F. Wotawa, and D. Li, *Software Fault Localization: an Overview of Research, Techniques, and Tools*, 2023.

[11] K. C. Youm, J. Ahn, J. Kim, and E. Lee, "Bug localization based on code change histories and bug reports," in *2015 Asia-Pacific Software Engineering Conference (APSEC)*, 2015, pp. 190–197.

[12] K. C. Youm, J. Ahn, and E. Lee, "Improved bug localization based on code change histories and bug reports," *Information and Software Technology*, vol. 82, 2017.

[13] H. L. Ribeiro, R. P. A. de Araujo, M. L. Chaim, H. A. de Souza, and F. Kon, "Jaguar: A spectrum-based fault localization tool for real-world software," in *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*, 2018, pp. 404–409.

[14] *Debugging Code*, 2019.