



# **Tracing the Trail: Investigating the Influence of Logging on Bug Resolution**

---

Amir Haghshenas

Naser Ezzati-Jivan  
Michel Dagenais

Winter 2024

# Agenda

---

- Problem definition
- Method
- Initial results
- Next steps

# Problem statement


---

- Developers usually follow an ad-hoc approach for logging a function.
- Goal of this track is to recommend whether a new function should be logged or not.
- Logging can also be done to achieve various goals. Such as “finding and fixing bugs”.
- This issue is also present for our Industry partners.
  - Currently discussing the issue with Matthew Khuozam and a line manager to present my work.
- We planned a two-phase approach to suggest logging decision for a new function.

Analyze the  
historical logging  
data


Learn from the  
extracted features

For a new  
function,  
suggest logging  
decision

- 
1. What is the relationship between bug-fixing commits and logging in terms of the quantity of logs, the levels of logs used, and modifications made to logs within those commits?
  2. What proportion of files that contain logging statements are involved in bug-fixing commits compared to those that are not? Does this suggest that logged files are more prone to bugs or more actively maintained?
  3. Is there a semantic relationship between log messages and bug-fix commit messages? Can we identify common themes or patterns in how developers describe issues in logs versus commit messages?

# Research Questions

---

- 
1. 10 high-rated Java projects in GitHub
    - a. Cassandra, Elasticsearch, Hadoop, spark and more.
  2. Trace Compass (Under analysis)
  3. Ericsson Private Project (under discussion)

The goal for this project selection is to cover both open-source community and the industry project to have a broader understanding.

# Project Selection

---

# Extracting bug-fixing commit



---

To extract bug-fixing commit, we rely on the commit message and issue reports and decide based on that.

- The commit mentioned a bug report id.
- The commit explained what was caused by the bug.
- Contains the combination of certain words (bug, problem, error and fix, solve, resolve)

We also tried LLM based approach.

- Used GPT-4o-mini API.
- Explained the same criteria.
- Did multiple round of prompt tuning.
- Manually evaluated the result.

# Extracting changed functions

---

- For each commit, we get all the changed files using GitHub API.
- For every changed file per commit, we downloaded the file before and after the change and save them locally.
- Using the file before the change, the file after the change and the diff file, we identified which functions are changed.
- A single file can be present in multiple commits, and we save all the files specific to each commit separately.
- The output of this step is a list of function name and file address.



1. Extracted two groups of features from each file.
  - a. Syntactic feature (Line of code, number of loops, number of branches, number of calls to other functions and more)
  - b. Logging features (Log level and log message for every log level present)
2. We extracted these features from both before the change and after the change files.

We first used LLM based approach for this feature extraction, but the result were not as good as we expect.

We switched to AST based method for extracting these features.

# Feature Extraction

---



# Preliminary Results

---

Only a small percentage of bug-fixing commits are logged

Changes in logging decision is more frequent in bug-fixing commit.

Bug-fixing commits on average have more severe log levels.

## Preliminary results

---

```

6a2ace739fa97dc25b41014944343d0c65e17c3a(10-2-2013)/castor/after_CastorMarshaller.java": {
  "description": "The function createXMLContext initializes a Castor XMLContext with mapping files, target classes, and target packages.",
  "features": {
    "before": {
      "number_of_loops": 3,
      "number_of_branches": 3,
      "number_of_lines_of_code": 27,
      "number_of_function_calls": 7,
      "logging_info": {
        "number_of_log_lines": 4,
        "log_details": [
          {
            "log_level": "info",
            "log_message": "Configured using [<mapping locations>]"
          },
          {
            "log_level": "info",
            "log_message": "Configured for target classes [<target classes>]"
          },
          {
            "log_level": "info",
            "log_message": "Configured for target packages [<target packages>]"
          },
          {
            "log_level": "info",
            "log_message": "Using default configuration"
          }
        ]
      }
    },
    "after": {
      "number_of_loops": 3,
      "number_of_branches": 5,
      "number_of_lines_of_code": 37,
      "number_of_calls_to_other_functions": 7,
      "logging_info": {
        "number_of_log_lines": 0,
        "log_details": []
      }
    }
  }
}

```

```

"e1720d89fcf65fca6b244df1696c1df67fc0c808(21-4-2014)/type/after_AnnotationMetadataTests.java": {
  "description": "The function asserts that the meta-annotations on a given AnnotationMetadata object have correctly overridden specified attributes.",
  "features": {
    "before": {
      "number_of_loops": 0,
      "number_of_branches": 5,
      "number_of_lines_of_code": 14,
      "number_of_calls_to_other_functions": 6,
      "logging_information": {
        "log_lines": 0,
        "log_details": []
      }
    },
    "after": {
      "number_of_loops": 0,
      "number_of_branches": 5,
      "number_of_lines_of_code": 22,
      "number_of_calls_to_other_functions": 5,
      "logging_information": {
        "number_of_log_lines": 5,
        "log_details": [
          {
            "log_level": "INFO",
            "log_message": "length of basePackages list"
          },
          {
            "log_level": "INFO",
            "log_message": "basePackages[0]"
          },
          {
            "log_level": "INFO",
            "log_message": "length of value list"
          },
          {
            "log_level": "INFO",
            "log_message": "length of 0th value array"
          },
          {
            "log_level": "INFO",
            "log_message": "length of basePackageClasses list"
          }
        ]
      }
    }
  }
}

```

1. To answer the third research question, we will use LLM-based approach to generate a semantic relation between log message, commit message and the potential issue report.
2. The tool will be available as a docker image for easy access.
3. Next phase of this research it to learn from the extracted features to suggest logging decision for a new function.

One solution fits all might not be possible. We are exploring providing a unique solution specific to a project based on learning from the history of that project.

## Next Steps

---



# Thank you

---

Amir Haghshenas

[Amir.Haghshenas@polymtl.ca](mailto:Amir.Haghshenas@polymtl.ca)