Replication of "Source Code Properties of Defective Infrastructure as Code Scripts" (Rahman & Williams, 2019)

1st MIGUEL CARRASCO GUIRAO
BARCELONA, SPAIN
miguel.carrasco-guirao@polymtl.ca

2nd POL MARGARIT FISAS
BARCELONA, SPAIN
pol.margarit-i-fisas@polymtl.ca

3rd ...
...,...
...polymtl.ca

4th ...
..., ...
...polymtl.ca

***Abstract*—This paper reports a replication of key parts of Rahman and Williams (2019). The replication focuses on (i) reproducing subsections 3.1.1 (Repository Collection) and 3.1.2 (Commit Message Processing) of the original methodology and (ii) answering research questions RQ1 and RQ3 using the authors' released datasets. We describe our mining setup with the GitHub API and caching to respect rate limits, operationalize commit message processing to build extended commit messages, and then use the supplied datasets to analyze source code properties of Puppet scripts (RQ1) and to build defect prediction models (RQ3). We report our findings, compare them against the original study, and discuss deviations and threats to validity.**

***Index Terms*—Replication, Mining Software Repositories, Infrastructure as Code, Puppet, Defect Prediction, Empirical Software Engineering**

I. Introduction

Infrastructure as Code (IaC) enables automated, reliable deployment pipelines. Defects in IaC scripts can undermine these pipelines. Rahman and Williams (2019) investigated which source code properties of Puppet scripts correlate with defectiveness and how those properties can be used to build defect prediction models.

In this project, we replicate part of their study by reproducing Sections 3.1.1 (Repository Collection) and 3.1.2 (Commit Message Processing), and by addressing two research questions: RQ1, which investigates what source code properties characterize defective IaC scripts, and RQ3, which explores how defect prediction models can be constructed using those properties. Our replication relies on the authors' released datasets and compares our findings against the original results.

II. Background

III. Research Questions

¡¡

IV. Methodology

This section describes the process followed to replicate the repository mining phase and implement the filtering restrictions defined by Rahman and Williams (2019). All steps were automated in Python 3 using the GitHub REST API, ensuring reproducibility and independence from manual dataset curation.

A. Repository Mining Setup

Repository data were collected automatically from GitHub. Each repository was queried via the REST API to obtain metadata and commit history. This approach was chosen to maintain transparency and to enable other researchers to replicate the same pipeline.

Only public repositories were considered (Restriction R1), as these allow full verification of the results and comply with open science principles. Private or archived repositories were excluded to ensure accessibility and data consistency.

B. Restriction R2 — Puppet File Ratio

To ensure that repositories were primarily Puppet-based, Restriction R2 required that at least 11% of the files in the repository be Puppet scripts (`.pp` files). The computation was based on the number of files rather than file size or lines of code. This decision was made for three reasons:

1) File counts are independent of script length, avoiding bias toward larger files that may not contain meaningful IaC logic.
2) Counting by files instead of bytes prevents the inclusion of large, non-source assets (e.g., binaries, documentation, or templates) that distort the ratio.
3) This method preserves alignment with the original study, which also defined R2 in terms of file counts.

For efficiency and to avoid GitHub API rate limits, repositories were cloned locally using a shallow clone (`git clone --depth 1`). This retrieves only the latest snapshot of the repository without full history, reducing both time and bandwidth while maintaining a complete view of the file structure.

C. Restriction R3 — Repository Activity

The original paper informally mentions "two commits per month", but the wording is ambiguous as to whether this must hold for every calendar month or only on average over a period. In this replication we make the assumption explicit and operationalize R3 as an *average* of at least two commits per month over the last 24 full months (excluding the current month), rather than enforcing the threshold for each individual month. This adjustment was made to reflect realistic development patterns in open-source projects, which

often experience uneven commit activity due to vacations, release cycles, or temporary inactivity.

The justification for this modification is threefold:

1) Commit activity is inherently bursty, and a strict monthly threshold may unfairly exclude active projects.
2) The average-based approach captures long-term project vitality while tolerating natural fluctuations.
3) Using a 24-month window provides a representative temporal scope of project activity without overweighting historical data.

The current month was excluded from calculations since it may contain incomplete data. Commits were collected via paginated API requests (100 commits per page) until reaching 24 months of history or the beginning of the repository's activity.

D. Commit Collection and Storage

For repositories satisfying R1–R3, all commits were extracted with their SHA, author date, and first-line message. These commits were serialized into a JSON structure and stored locally for reproducibility. This design allows later verification of data integrity and potential reuse for further analysis, such as the classification of defect-inducing commits.

E. Analysis Scope

While the mining pipeline successfully reproduces the filtering and collection logic of Rahman and Williams (2019), the datasets used to address RQ1 and RQ3 were obtained directly from the authors' released data. This ensures that the statistical and machine learning analyses are consistent with the original study while verifying the reproducibility of the mining process itself. The data collected through our own mining implementation were not used in the analyses, as this step was purely formative and optional, and would have additionally required extensive preprocessing and message filtering to match the original dataset's structure.

F. Defect Prediction Models (RQ3)

V. Results

A. Repository Mining Phase

The mining phase was executed using the Python pipeline described in the previous section. Initially, the repositories referenced in Rahman and Williams (2019) were analyzed, specifically the seven OpenStack Puppet projects originally included in their study:

- openstack/puppet-keystone
- openstack/puppet-nova
- openstack/puppet-neutron
- openstack/puppet-glance
- openstack/puppet-cinder
- openstack/puppet-horizon
- openstack/puppet-swift

The following execution summary was obtained when running the mining script:

```
Checking openstack/puppet-keystone...
openstack/puppet-keystone: <11% of .pp
Checking openstack/puppet-nova...
openstack/puppet-nova: <11% of .pp
Checking openstack/puppet-neutron...
openstack/puppet-neutron: <11% of .pp
Checking openstack/puppet-glance...
Checking openstack/puppet-cinder...
Checking openstack/puppet-horizon...
openstack/puppet-horizon: <11% of .pp
Checking openstack/puppet-swift...
Commits saved in output/mined_commits.json
```

From this execution, only `openstack/puppet-swift` met all restrictions (R1–R3) and was successfully mined. The rest were excluded because they no longer satisfy Restriction R2 (less than 11% of files are Puppet scripts) or, in some cases, exhibit very low or irregular commit activity that violates Restriction R3.

In addition, we attempted to reproduce the mining process using the alternative repositories provided in the course materials (e.g., `mozilla/gecko-dev`, `wikimedia/mediawiki`, `Mirantis/kubernetes-release`), but none of these projects satisfied the Puppet ratio or activity thresholds either. These repositories use different configuration management systems or contain mixed languages, leading to automatic exclusion by our filters.

This confirms that, while the mining pipeline itself is functional and consistent with the logic of Rahman and Williams (2019), most of the original and educational repositories are no longer valid candidates for replication due to ecosystem changes.

B. Defect Prediction Models (RQ3)

VI. Discussion

A. Repository Mining Discussion

The mining experiment reveals that most repositories originally analyzed by Rahman and Williams (2019) no longer comply with the same restrictions (R2 and R3). The main causes are: (1) project archival or migration from Puppet to newer frameworks such as Ansible or YAML-based pipelines, (2) a significantly reduced proportion of Puppet files in mixed-language repositories, and (3) decreased development activity leading to violation of the minimum commit frequency requirement.

Moreover, the repositories provided as examples for the course exercise (e.g., Mozilla, Wikimedia, and Mirantis) also failed to meet the criteria, either because they do not primarily use Puppet or because their activity levels are inconsistent with the R3 threshold. This further emphasizes the difficulty of reproducing historical studies when the technological landscape has evolved.

The decision to measure the Puppet ratio (R2) by file count rather than file size proved reliable and allowed for objective filtering, although the 11% threshold is increasingly restrictive

in modern multi-language repositories. The adaptation of Restriction R3 to an average-based metric was also validated, as open-source projects typically display bursty commit behavior with periods of inactivity.

Overall, the mining phase successfully replicates the filtering logic of the original study but shows that reproducing the exact dataset is no longer feasible. The only repository satisfying all constraints, `openstack/puppet-swift`, confirms that the implemented restrictions and mining logic behave correctly, even if the underlying ecosystem has evolved.

B. Defect Prediction Models Discussion

VII. Conclusion

Acknowledgment

References

[1] M. Rahman and L. Williams, "Source Code Properties of Defective Infrastructure as Code Scripts," in *Proceedings of the 2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2019, pp. 246–256.

[2] GitHub REST API Documentation. Available: https://docs.github.com/en/rest

[3] Puppet Documentation. Available: https://puppet.com/docs/puppet/latest/puppet_index.html