Quality Assurance for Infrastructure Orchestrators: Emerging Results from Ansible

Yue Zhang[*] Muktadir Rahman[†] Fan Wu[‡] Akond Rahman[§]
[*]Department of Computer Science, Tuskegee University, Tuskegee, AL, USA
[†]MetaDesign Solutions, Dhaka, Bangladesh
[‡]Department of Computer Science, Tuskegee University, Tuskegee, AL, USA
[§]Department of Computer Science and Software Engineering, Auburn University, Auburn, AL, USA
Email: [*]yzhang8317@tuskegee.edu [†]muktadir.rahman@mds.com.bd [‡]fwu@tuskegee.edu [§]akond@auburn.edu

***Abstract*—Infrastructure as code (IaC) is the practice of automatically managing computing infrastructure at scale. Despite yielding multiple benefits for organizations, the practice of IaC is susceptible to quality concerns, which can lead to large-scale consequences. While researchers have studied quality concerns in IaC manifests, quality aspects of infrastructure orchestrators, i.e., tools that implement the practice of IaC, remain an under-explored area. A systematic investigation of defects in infrastructure orchestrators can help foster further research in the domain of IaC. From our empirical study with 22,445 commits mined from the Ansible infrastructure orchestrator we observe (i) a defect density of 17.9 per KLOC, (ii) 12 categories of Ansible components for which defects appear, and (iii) the 'Module' component to include more defects than the other 11 components. Based on our empirical study, we provide recommendations for researchers to conduct future research to enhance the quality of infrastructure orchestrators.**

***Index Terms*—ansible, devops, infrastructure as code**

I. Introduction

Infrastructure as code (IaC) is the practice of automatically managing computing infrastructure at scale [7] using a state reconciliation approach. IaC is implemented using infrastructure orchestrators, i.e., tools that account for state reconciliation to implement the practice of IaC, such as Ansible. Use of infrastructure orchestrators have yielded several benefits for organizations. Along with automation, the practice of IaC recommends application of quality assurance activities, such as static analysis and testing. Infrastructure orchestrators are no different: quality assurance activities also need to be applied for infrastructure orchestrators. However, quality assurance of infrastructure orchestrators remain an under-explored area. A systematic investigation of defects that occur for infrastructure orchestrators can lay the groundwork for future research.

To that end, we conduct a preliminary empirical study where we analyze 22,445 commits mined from the repository of the Ansible infrastructure orchestrator [1]. We apply a qualitative analysis technique called open coding [15] using which we quantify defect frequency, and derive component categories for which defects appear. Dataset and source code used in our empirical study are available online [2]. With our empirical study, we answer the following research questions: (i) ***RQ1*: How frequently do defects occur for the Ansible infrastructure orchestrator?**, and (ii) ***RQ2*: In what components do defects appear for the Ansible infrastructure orchestrator?**

Our Contribution is an empirical analysis of defects in components within the Ansible infrastructure orchestrator.

II. Related Work

Our paper is related to prior research that have investigated quality assurance for Ansible. Dalla Palma et al. [4] introduced a set of 46 metrics that can be used to predict defective Ansible manifests. In another paper, Dalla Palma et al. [3] found code-based metrics to be better than process metrics for defect prediction for Ansible manifests. Opdebeeck et al. [12] identified code smells that can cause defects in Ansible manifests. Hassan and Rahman [11] categorized defects observed in Ansible test manifests. Kokuryo et al. [9] observed execution of external manifests as a quality concern in Ansible manifests. Specific categories of defects, such as security defects, have also garnered interest. Rahman et al. [14] derived a taxonomy of security defects in Ansible manifests. Rahman et al. [14]'s paper was replicated by Hortlund [6], who reported the security weakness density to be less than that reported by Rahman et al. [14]. From the afore-mentioned discussion, we observe a lack of research on how frequently defects occur for the Ansible infrastructure orchestrator. We address this research gap in our paper.

III. Methodology

A. Methodology for RQ1

We conduct our empirical study by mining the OSS repository for the Ansible infrastructure orchestrator [1]. We download this repository with 52,245 commits on May 01, 2022. We mine commits from this repository, and the corresponding Python files that are modified in each commit. We select Python files as it is the primary language to implement the Ansible infrastructure orchestrator [1].

Keyword search: We apply a keyword search to identify commit messages similar to prior defect categorization research [5], [13]. The keywords are: 'bug', 'defect', 'error', 'fault', 'fix', 'flaw', 'incorrect', 'issue', and 'mistake'. Using our keyword search we identify 22,445 commits.

Qualitative analysis: We apply a qualitative analysis technique called closed coding [15] with commit messages and corresponding diffs from the set of 22,445 commits. For each of these commits and their corresponding diffs, two raters individually identify if a defect appears in the commit. To identify defects, we use the IEEE definition [8]: "*an imperfection or deficiency in the code that needs to be repaired*". A rater determines (i) if problematic code exists in the commit, (ii) if problematic code leads to an immediate incorrect or undesired consequence upon execution that is explicitly expressed, and (iii) if the problematic code was repaired. The first author and a PhD student in the department are raters. We calculate Krippendorff's α [10] α is 0.52, indicating 'unacceptable' agreement [10]. Both raters discussed their disagreements, and upon discussion, they conduct the inspection process again. At this stage, we calculate Krippendorff's α to be 1.0, indicating 'perfect' agreement [10]. From open coding, we obtain a mapping between each commit and whether the commit is defect-related. From this mapping we also identify which Python files are modified in a commit that is defect-related. We refer to this set of files as defect-related Python files.

Metrics: We use two metrics: (i) defect proportion, and (ii) defect density. Defect proportion quantifies the proportion of defect-related commits. Defect density corresponds to count of defects that appear in every 1,000 lines of Python code.

B. Methodology for RQ2

RQ2 focuses on identifying the components in the Ansible infrastructure orchestrator for which defect appears. We use the Python files that are modified in a defect-related commit from Section III-A to derive the components. *First*, from the collected Python files from Section III-A, we exclude files that are not part of the orchestrator, such as testing-related files. *Second*, we extract the file names for each Python file. Third, we apply open coding [15] to derive component categories based on the similarities between names.

The open coding process is conducted by the first author, which is susceptible to rater bias. We mitigate this limitation by using another rater who is the last author. We calculate Krippendorff's α to be 0.69 between the first and last author, indicating 'acceptable' agreement [10]. Upon completion of the open coding process, we obtain a mapping between each of the Python files obtained from Section III-B and a component category. We use this mapping to quantify the frequency of each component category by reporting the proportion of defect-related Python files that map to each identified category.

IV. Empirical Findings

We provide answers to our research questions as follows:

A. Answer to RQ1

In this section, we answer **RQ1: How frequently do defects occur for the Ansible infrastructure orchestrator?** We identify 4,554 defect-related commits. From Table I, we observe defect density and defect proportion to respectively, be 17.9 and 30.3%. From identified 4,554 defect-related commits we observe 14,756 Python files to be modified.

TABLE I: Answer to RQ1: Frequency of defects

Metric	**Value**
Defect density	17.9 (per KLOC)
Defect proportion	30.3 %

B. Answer to RQ2

In this section, we answer **RQ2: In what components do defects appear for the Ansible infrastructure orchestrator?** using Table II. The 'Definition' and 'Frequency' columns respectively provides the definitions and proportion of defect-related Python files that map to each identified category. For example, 0.06% of the defect-related Python files belong to 'Cache'. For 'Module', we identify 10,215 defect-related Python files amongst which 78.6%, 4.9%, and 16.5% are the types of builtin, core, and plugin.

V. Discussion and Conclusion

We discuss our findings in the following subsections:

A. Implications Related to Future Work

Our empirical findings lay the groundwork for future research:

Construct Defect Taxonomies: Table I shows that defects are prevalent, e.g., 20.3% of the 22,445 commits are labeled as defective from our qualitative analysis. As defects are prevalent in the Ansible infrastructure orchestrator, we recommend researchers to gain a systematic understanding of defect categories to develop a comprehensive taxonomy that will identify defects unique to IaC's state reconciliation approach. Such taxonomy can help the IaC community to advance the science of IaC quality assurance, as well as develop automated program repair techniques.

Develop Testing and Verification Techniques: Our results show that the Ansible infrastructure orchestrator contains multiple components, each of which performs a distinct operation. Also, results reported in Table II show that defect-related Python files are not uniformly distributed across the identified components. We hypothesize that components for which defect-related files appear more can be prioritized for performing testing and verification. Furthermore, we hypothesize one technique to not discover latent defects with equal efficacy for all identified components. Future research can support or refute our hypotheses.

Limitations: We discuss the limitations of our paper below:

Construct Validity: The raters may have implicit biases that could have affected the labeling process described in Section III. We mitigate this limitation by using two raters. Furthermore, we use commit messages to identify defects that may not capture the full context of defects for the Ansible infrastructure orchestrator.

External Validity: Our empirical study is susceptible to external validity as we only use the Ansible OSS repository.

TABLE II: Answer to RQ2: Name, Definition, and Frequency of Components for Which Defects Appear

Name	Definition	Frequency
Cache	This component pre-fetches infrastructure-related data so that infrastructure operations are performant.	0.06%
Command Line Interface (CLI)	This component is responsible for running a single task on desired computing infrastructure.	2.39%
Collection	This component is responsible to manage Ansible-specific collections, such as roles and plugins.	0.01%
Compatibility	This component facilitates execution of Ansible playbooks that are developed using multiple versions of Ansible.	0.06%
Error Management	This component handles and reports errors, such as non-zero return codes, which are generated by the Ansible runtime.	0.01%
Executor	This component receives infrastructure configurations parsed from Ansible playbooks, and decides what to execute next.	6.23%
Galaxy	This component manages pre-packaged units of Ansible playbooks, which are called 'roles'. Ansible Galaxy allows practitioners to find roles, use existing roles with the `ansible-galaxy` command, allow practitioners to create roles, and rank roles based on the user feedback.	0.41%
Inventory	This component manages the computing infrastructure that is specified as inventory files written in INI and YAML.	1.98%
Module	This component contains modules that practitioners can use to perform infrastructure-related tasks. Ansible provides two categories of modules: (i) builtin modules, which are code units that perform specific functions, and users can call existing modules locally or remotely to execute automation tasks without needing all the details; and (ii) plugins that extend the functionality of Ansible, but can only be executed on the machine running ansible and not the remote machine.	69.23%
Parsing	This component parses Ansible playbooks written in YAML. Output of this module is later used by the executor component of Ansible.	0.94%
Play Management	This component determines when and where tasks for each playbooks are completed and by whom.	3.67%
Utils	This component contains source code that is used to perform utility-based operations while parsing and executing playbooks.	15.00%

B. Conclusion

Infrastructure orchestrators play a pivotal role in automated infrastructure management, which necessitates integration of quality assurance activities for infrastructure orchestrators. We have conducted an empirical study with the Ansible infrastructure orchestrator, for which we have observed a defect density of 17.9 per KLOC. We also identify 12 component categories within Ansible, amongst which the 'Module' component contains the most amount of defect-related Python files. Based on our findings, we recommend researchers to (i) construct defect taxonomies unique to infrastructure orchestrators, (ii) develop techniques that will test and verify infrastructure orchestrator components with prioritization heuristics, and (iii) quantify the correlation between maintenance efforts and quality assurance concerns for infrastructure orchestrators.

ACKNOWLEDGMENTS

We thank the PASER group at Auburn University for their valuable feedback. This research was partially funded by the U.S. National Science Foundation (NSF) Award # 2247141, Award # 2310179, Award # 2209637, and the U.S. National Security Agency (NSA) Award # H98230-21-1-0175. We also thank Farhat Lamia Barsha for her help with the qualitative analysis.

REFERENCES

[1] ansible, "ansible/ansible," https://github.com/ansible/ansible, 2022, [Online; accessed 02-Dec-2022].

[2] A. Authors, "Verifiability package for paper," https://figshare.com/s/32b4124f7b99ca521cc0, 2022, [Online; accessed 01-Dec-2022].

[3] S. Dalla Palma, D. Di Nucci, F. Palomba, and D. A. Tamburri, "Within-project defect prediction of infrastructure-as-code using product and process metrics," *IEEE Transactions on Software Engineering*, vol. 48, no. 6, pp. 2086–2104, 2022.

[4] ——, "Toward a catalog of software quality metrics for infrastructure code," *Journal of Systems and Software*, vol. 170, p. 110726, 2020.

[5] J. Garcia, Y. Feng, J. Shen, Y. Almanee, Sumaya Xia, and Q. A. Chen, "A comprehensive study of autonomous vehicle bugs," in *Proceedings of the 42nd International Conference on Software Engineering*, ser. ICSE '20, 2020.

[6] A. Hortlund, "Security smells in open-source infrastructure as code scripts: A replication study," 2021.

[7] J. Humble and D. Farley, *Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation*, 1st ed. Addison-Wesley Professional, 2010.

[8] IEEE, "IEEE standard classification for software anomalies," *IEEE Std 1044-2009 (Revision of IEEE Std 1044-1993)*, pp. 1–23, Jan 2010.

[9] S. Kokuryo, M. Kondo, and O. Mizuno, "An empirical study of utilization of imperative modules in ansible," in *2020 IEEE 20th International Conference on Software Quality, Reliability and Security (QRS)*, 2020, pp. 442–449.

[10] K. Krippendorff, *Content analysis: An introduction to its methodology*. Sage publications, 2018.

[11] H. Mohammad Mehedi and A. Rahman, "As code testing: Characterizing test quality in open source ansible development," in *2022 15th IEEE Conference on Software Testing, Verification and Validation (ICST)*. Los Alamitos, CA, USA: IEEE Computer Society, apr 2022.

[12] R. Opdebeeck, A. Zerouali, and C. De Roover, "Smelly variables in ansible infrastructure code: Detection, prevalence, and lifetime," in *2022 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*. IEEE, 2022.

[13] A. Rahman, E. Farhana, C. Parnin, and L. Williams, "Gang of eight: A defect taxonomy for infrastructure as code scripts," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, ser. ICSE '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 752–764.

[14] A. Rahman, M. R. Rahman, C. Parnin, and L. Williams, "Security smells in ansible and chef scripts: A replication study," *ACM Trans. Softw. Eng. Methodol.*, vol. 30, no. 1, Jan. 2021.

[15] J. Saldaña, *The coding manual for qualitative researchers*. Sage, 2015.