# A Framework for Measuring the Quality of Infrastructure-as-Code Scripts

Pandu Ranga Reddy Konala<sup>†</sup>, Vimal Kumar<sup>†</sup>, David Bainbridge<sup>†</sup>, Junaid Haseeb<sup>†</sup>

School of Computing and Mathematical Sciences, University of Waikato

Hamilton, New Zealand 3240

{pkonala, vkumar, davidb, jhaseeb}@waikato.ac.nz

Abstract—Infrastructure as Code (IaC) has become integral to modern software development, enabling automated and consistent configuration of computing environments. The rapid proliferation of IaC scripts has highlighted the need for better code quality assessment methods. This paper proposes a new IaC code quality framework specifically showcased for Ansible repositories as a foundation. By analyzing a comprehensive dataset of repositories from Ansible Galaxy, we applied our framework to evaluate code quality across multiple attributes. The analysis of our code quality metrics applied to Ansible Galaxy repositories reveal trends over time indicating improvements in areas such as metadata and error handling, while highlighting declines in others such as sophistication and automation. The framework offers practitioners a systematic tool for assessing and enhancing IaC scripts, fostering standardization and facilitating continuous improvement. It also provides a standardized foundation for further work into IaC code quality.

**Keywords:** Infrastructure as Code, Code Quality Framework, Ansible, IaC Scripts, Configuration Management, Software Quality, Code Quality, Software Maintainability, Code Standardization, Repository Analysis, DevOps Practices, Automation

#### I. INTRODUCTION

Infrastructure as Code (IaC) has revolutionized IT infrastructure management by treating configurations as version-controlled, executable code, enabling automation, scalability, and consistency. However, as IaC becomes integral to software development, specific quality attributes are needed, since traditional standards like ISO/IEC 25010:2011 (Systems and software Quality Requirements and Evaluation — SQuaRE) [1] are designed for application software and do not fully address the unique characteristics of IaC.

In this paper, we introduce an IaC quality framework consisting of nine categories, as presented herein, aligned with the aforementioned ISO/IEC 25010:2011 standard. This framework builds on and differentiates from prior work by Rahman and Williams [2], Dalla Palma et al. [3], and Begoug et al. [4] by tailoring the categories to the specific context of IaC. These categories—Metadata, Code Structure, Code Sophistication, Code Maintainability, Code Security, Function and Purpose, Error Handling, Automation, and Code Integration—provide a comprehensive framework that encompasses all identified IaC code attributes while remaining grounded in established principles.

We validated the framework through empirical analysis of repositories from Ansible Galaxy [5], which highlighted notable trends in IaC script quality. This analysis affirms the relevance of the proposed categories and demonstrates its practical applicability in evaluating real-world IaC scripts.

We present this investigation into the relationship between the attributes and IaC quality assessment practices in the form of five key research questions (RQs).

- **RQ1:** What mechanisms are currently employed by IaC tools to evaluate the quality of their scripts, and how consistent are these methods across the different tools?
- RQ2: How do existing IaC code quality attributes map to the characteristics defined in the ISO/IEC 25010:2011 standard, and what are the limitations of this mapping?
- RQ3: Which specific IaC code quality categories can be developed to accommodate all identified IaC code attributes while drawing inspiration from ISO/IEC 25010:2011?
- RQ4: How does the proposed IaC code quality framework perform when applied to real-world IaC repositories, and what insights can be gained?
- **RQ5:** In what ways can the proposed IaC code quality framework enhance the assessment of IaC scripts in practice?

# II. BACKGROUND

This section establishes a foundation for analyzing IaC quality, focusing on tools for managing IT infrastructures. It compares IaC tools such as Ansible, Terraform, Chef, and Puppet, examining their features and assessing their suitability in various contexts. By evaluating their strengths and limitations, the section provides insights into tool usage and discusses methodologies for code quality evaluation and assurance. It also compares frameworks and scoring mechanisms to explore their influence on IaC quality management.

IaC is categorized into three areas [6] that automate IT processes: *Infrastructure Provisioning* automates allocation and management of hardware resources, with tools like Terraform and AWS CloudFormation [7] enabling resource

deployment. Configuration Management automates software infrastructure configuration; tools such as Ansible, Chef, and Puppet codify system configurations, reducing errors and enhancing reproducibility. Image Building standardizes machine and container images; tools like Packer [8] create machine images, while Docker [9] enables containerization for application portability. In summary, IaC tools automate provisioning, configuration, and image building, improving efficiency and reliability in IT operations.

#### A. COMPARING EXISTING IAC TOOLS AND PRACTICES

• **RQ1:** What mechanisms are currently employed by IaC tools to evaluate the quality of their scripts, and how consistent are these methods across the different tools?

IaC tools employ mostly rudimentary mechanisms to evaluate script quality, but these mechanisms also differ across platforms and categories, indicating a lack of standard assessment methodologies as shown in Table I. For instance, repositories such as Ansible Galaxy, Puppet Forge, and Chef Supermarket implement quality control measures, but their implementation and transparency vary. Configuration management tools incorporate custom scoring systems to ensure quality control in maintaining IaC repositories. Notably, infrastructure provisioning and image building tools typically lack these mechanisms or have assessments that are less common.

The analysis of quality scoring systems across various IaC platforms, detailed in Table II, reveals heterogeneity in assessment practices and a lack of collective standardization. This variation poses challenges for comparability and consistency, potentially impacting user trust and assessment reliability. This highlights the need for standardized, transparent, and comprehensive quality metrics for IaC tools. Implementing a unified quality assessment approach would facilitate informed decision-making and enhance the reliability and efficiency of the IaC ecosystem.

#### B. RELATED WORK

IaC has transformed the management and provisioning of IT infrastructure; however, maintaining the quality and standardization of IaC scripts continues to be a critical challenge. A survey by Konala et al. [24] highlights that several tools exist that are designed to address specific quality issues such as security smells or code smells through the use of various techniques like regular expressions, linting, deep learning, etc. Despite the advancements made with these tools, our analysis reveals a lack of quality assessment approaches to evaluating overall IaC code quality. This gap underscores the need for a higher-level solutions that evaluate IaC scripts.

Most existing work addresses this research gap by proposing quality attributes and metrics tailored to different IaC tools. Rahman et al. [2] initially conducted an empirical study on Puppet scripts, identifying 12 source code properties correlated with defects, such as lines of code and hard-coded strings. Their defect prediction models achieved precision between

0.70–0.78 and recall between 0.54–0.67, offering practitioners actionable insights to enhance script quality. However, the study's focus on Puppet scripts limits its applicability to other IaC tools like Ansible, Chef, or Terraform. Additionally, the lack of alignment with ISO standards suggests the need for work to make these attributes more universally applicable.

Similarly, Dalla Palma et al. [3] developed a catalog of 46 quality attributes specific to Ansible scripts, addressing aspects such as code complexity, maintainability, error handling, and best practices. While beneficial for Ansible users, these attributes need adaptation and validation to apply across other IaC environments like Chef or Terraform. Additionally, their impact on quality outcomes has yet to be empirically validated. Begoug et al. introduced TerraMetrics [4], a tool for measuring Terraform script quality using 40 attributes based on static code analysis which draws parallels to Dalla Palma et al. [3] study. Despite its focus on code quality, its limited validation on only three repositories highlights the need for broader assessment. Overall, the previous work in this area points to a need for standardized quality assessment that can be applied across diverse IaC tools.

➤ RQ1 Takeaway Message: IaC tools use wide range of methods to evaluate script quality, creating challenges in ensuring consistency and comparability across platforms.

# C. ABOUT ISO/IEC 25010:2011 SOFTWARE QUALITY STANDARDS

ISO/IEC 25010:2011, part of the Systems and Software Quality Requirements and Evaluation (SQuaRE) model, is an international standard for assessing software and system quality. It builds on the ISO/IEC 9126 [25] standard by introducing ten quality characteristics that help organizations evaluate key aspects such as functionality and reliability. The framework consists of two models: the Quality in Use Model, which includes five characteristics related to user interaction applicable to human-computer interfaces, and the Product Quality Model, which encompasses eight characteristics addressing the static and dynamic aspects of software and computer systems. In the context of IaC and this research, the Product Quality Model is particularly relevant. IaC involves managing and provisioning infrastructure through machine-readable definition files, which aligns with the static and dynamic properties covered by the model's eight characteristics. Applying the Product Quality Model to IaC ensures that infrastructure provisioning meets operational and quality standards.

In terms of relating the work presented here to ISO/IEC 25010:2023 [26]—the relevant part of the successor to ISO/IEC 25010:2011, but at the time of writing yet to confirmed as a standard—there are two key areas of change. The first is that "usability" has been renamed to "interaction capability" and "portability" to "flexibility," which are cosmetic changes. The second is the inclusion of an

TABLE I IAC SOFTWARE TOOLS REPOSITORY & QUALITY SCORING METRICS

IaC Categories	IaC Software Tools	Repository	Quality Scoring Metrics
	Terraform [10]	Terraform Registry [11]	None
Infrastructure Provisioning	CloudFormation (AWS)	None	None
	Azure Resource Manager (ARM) Templates [12]	Azure Quickstart Templates [13]	Pass/Fail
	Google Cloud Deployment Manager [14]	None	None
Configuration Management	Ansible [15]	Ansible Galaxy	Scale of 0-5 points
	Puppet [16]	Puppet Forge [17]	Scale of 0-100%
	Chef [18]	Chef Supermarket	Scale of 0-100%
	SaltStack (Salt) [19]	SaltStack Formula [20]	None
Image Building	Packer	None	None
	Docker	Docker	Verified Publisher Program

TABLE II Comparison of Quality Scoring Systems across Ansible Galaxy, Puppet Forge, Chef Supermarket, Azure, and Docker

Feature	Ansible Galaxy	Puppet Forge	Chef Supermarket	Azure	Docker
Scoring Method	Weighted combination of community and quality scores, based on user surveys.	Automated evaluation for code formatting and malware checks.	Community-defined quality metrics, including publication status and platform support.	Compliance checks against Azure best practices.	Docker Security Scanning and Docker Verified Publisher Program.
Scoring Scale	Point based scoring ranging from 0 to 5.0 for both Community and Quality Scores.	Percentage based scoring.	Percentage based scoring.	Binary Pass/Fail assessment.	None
Evaluation Tools	yamllint [21], ansible-lint [22], import-time checks.	VirusTotal for binary security checks.	Foodcritic [23] for cookbook evaluation.	Azure Policy, ARM Template Test Toolkit.	Docker Scout for vulnerability analysis.
Score Components	Community input, Syntax Score, Metadata Score.	Code formatting, security checks.	Publication status, collaboration, licensing, platform support.	Compliance with Azure standards, security checks, resource optimization.	Security vulnerabilities, image integrity, publisher verification.
Critique Points	User survey dependence, temporal limitation, tool efficacy.	Lack of detailed documentation, focus on code and malware only.	Emphasis on objective metrics, late-stage evaluation, community feedback reliance.	Limited visibility into detailed scoring criteria.	Limited visibility into detailed scoring criteria.

additional characteristic: "safety." This covers aspects such as failsafe mechanisms and operational constraints. As current IaC tools provide limited support for this, which itself is through 3rd-party plugins, this was not included in the analysis reported here. We note however, that it would be a straightforward matter to reapply the methodology to include this additional characteristic, in response to safety feature support becoming mainstream in IaC tools.

#### D. RESEARCH GAPS IN STATE-OF-THE-ART STUDIES

Despite advancements in assessing IaC quality, significant gaps persist due to the lack of standardized calculations and theoretical foundations. Much of the existing research is tool-specific, resulting in fragmented approaches that limit the generalizability of findings across different IaC platforms. This fragmentation hinders the development of a robust framework for evaluating IaC code quality, which is essential for ensuring consistent assessments.

Rahman et al.'s [2] study on Puppet scripts, for instance, highlights code properties linked to defects but is constrained by its narrow focus on Puppet and its non-alignment

with international software quality standards like ISO/IEC 25010:2011. Similarly, Dalla Palma et al.'s [3] work on Ansible scripts catalogues 46 quality attributes but lacks empirical validation and standardization like Rahman et al.'s work. Tools like TerraMetrics [4], designed for Terraform, also suffer from limited validation, small sample sizes, and non-adherence to recognized standards, raising concerns about their broader applicability.

Overall these approaches primarily focus on identifying low-level issues such as security smells or code smells without providing an evaluation of overall code quality. Several IaC tool repositories provide their own quality scoring mechanisms. The lack of thorough documentation about these mechanisms however, creates a transparency issue in addition to the issue of consistency as each platform employs its own mechanism (see Table II). This underscores the need for a standardized and empirically-backed framework, to improve IaC quality assessment and support the reliability of modern IT infrastructures.

# III. METHODOLOGY FOR MAPPING IAC CODE ATTRIBUTES

In IaC, attributes serve as fundamental components of configuration scripts. Defined as key-value pairs, these attributes convey information about configurations, target systems, permissions, and conditions necessary for infrastructure setup and management. Attributes direct the execution of IaC scripts, detailing the specific tasks to be performed, on which systems, and with what permissions. Analyzing such attributes for their alignment with quality characteristics, provides a basis for assessing their suitability in meeting quality requirements.

To illustrate, consider a basic Ansible playbook snippet [15] that installs Nginx:

#### Example Ansible Script for Installing Nginx

- name: Install Nginx
hosts: web\_servers
become: yes

tasks:

- name: Install Nginx

apt:

name: nginx state: present

In the above example, each attribute is described based on definitions from the Ansible documentation [27], contextualized for this specific case:

- The **name** attribute specifies the purpose of the play, here defined as "Install Nginx."
- hosts designates the target machines for task execution, in this case, a group labeled web\_servers.
- **become** enables privilege escalation, allowing the task to execute with elevated permissions.
- tasks lists actions to perform on the target servers. Each entry in the tasks list represents a specific task that Ansible will carry out.
- apt specifies the module used in this task, in this case, managing packages on Debian-based systems.
- **state** is a key within the **apt** module that defines the desired state of the package. Setting **state**: present ensures that Nginx will be installed if it is not already on the system.

These attributes define the operational parameters of the playbook, establishing its scope, permissions, and task requirements. The usage of attributes in a script such as the one above can be mapped to quality characteristics such as in ISO/IEC 25010:2011. Building on this foundation, the following section describes the methodology developed to address code quality issues in IaC scripts, structured into three key steps.

- Step 1: Mapping IaC Code Attributes to ISO/IEC 25010:2011. This initial mapping examines the extent to which the ISO/IEC 25010:2011 standard can accommodate specific IaC code attributes.
- Step 2: Development and Mapping of IaC Code Attributes to the Proposed IaC Code Quality Framework. This step assesses how the proposed framework meets the unique requirements of IaC, especially where ISO/IEC 25010:2011 may have limited applicability.
- Step 3: Comparison of Mapping Between ISO/IEC 25010:2011 and the Proposed IaC Code Quality Framework. This final step presents a comparative analysis, demonstrating the proposed framework's capacity to capture IaC-specific characteristics in comparison with the ISO/IEC 25010:2011 standard.

STEP 1: MAPPING IaC CODE ATTRIBUTES TO ISO/IEC 25010:2011

• **RQ2:** How do existing IaC code quality attributes map to the characteristics defined in the ISO/IEC 25010:2011 standard, and what are the limitations of this mapping?

Building on Rahman et al.'s [2] identification of 12 source code properties/attributes indicating defects in Puppet scripts and Dalla Palma et al.'s [3] expansion to 46 attributes in Ansible scripts, we assesses whether the ISO/IEC 25010:2011 framework is able to capture IaC's quality features. The quality attributes from previous work were categorized relative to ISO/IEC 25010:2011 based on their definitions into direct mapping, indirect mapping, and no mapping. Direct mapping includes IaC properties that correspond to ISO sub-characteristics and attributes, indirect mapping includes attributes that align operationally with ISO sub-characteristics without direct correspondence, and no mapping includes attributes unique to IaC that do not fit within the ISO framework.

During the mapping source properties identified by Rahman et al., attributes such as "Hard-Coded String" and "Lines of Code" were indirectly mapped to ISO/IEC 25010:2011 due to their impact on security and portability, respectively, while the "URL" metric had no mapping as it does not align with the standard's quality characteristics. Table III, shows the complete mapping. Similarly, Dalla Palma et al.'s analysis of 46 Ansible script attributes features like "Variables," "Regex," "Conditions," "Decisions," and "User Interactions" as indirectly mapped to ISO characteristics such as functional suitability, performance efficiency, and usability, and attributes like "Suspicious Comments" and "Deprecated Keywords and Modules" were linked to security concerns. Attributes such as "Plays," "Roles," and "Filters" had no direct mapping due to their unique roles within IaC. Table IV shows the complete mapping.

In both tables it can be seen that there are source properties that cannot be mapped to any ISO characteristic. As the number of properties increase in a given quality measurement system, the number of properties that can't be mapped to any ISO characteristic will also increase. This demonstrates the limitations of the ISO standard in fully capturing IaC-specific quality attributes and highlights the need for a framework tailored to evaluate IaC code quality effectively. It should be noted that Begoug et al.'s [4] work was excluded from this mapping given their alignment with Dalla Palma et al.[3] for simplicity.

➤ RQ2 Takeaway Message: Although certain IaC code quality attributes correspond directly or indirectly to the characteristics of ISO/IEC 25010:2011, many fall outside its scope, revealing limitations in the standard's ability to capture IaC-specific code quality features.

TABLE III
MAPPING OF IAC CODE ATTRIBUTES OF RAHMAN ET AL.'S [2] STUDY TO
ISO/IEC 25010:2011

ISO 25010:2011 Characteristics	Direct Mapping	Indirect Mapping
Functional Suitability	Include, Ensure, Require, Attribute	×
Performance Efficiency	×	×
Compatibility	×	X
Usability	Comment	X
Reliability	Ensure, File Mode, File	×
Security	SSH Key, File Mode	Hard-Coded String
Portability	Comment	Lines of Code
Maintainability	File	X

No Mapping
URL, Command

#### Legend:

X - No IaC code attributes maps to ISO 25010:2011 characteristics

STEP 2: DEVELOPMENT AND MAPPING OF IAC CODE ATTRIBUTES TO THE PROPOSED IaC CODE QUALITY FRAMEWORK

• **RQ3:** Which specific IaC code quality categories can be developed to accommodate all identified IaC code attributes while drawing inspiration from ISO/IEC 25010:2011?

While ISO/IEC 25010:2011 offers a comprehensive model for software product quality, the previous analysis through the mappings shows that there is a need to adapt this standard to address the unique aspects of IaC. We propose a framework consisting of a set of quality categories that capture the uniqueness of IaC code as opposed to general software. This framework is compatible with the ISO standard as we show that the quality categories can be mapped to the the quality characteristics in the standard.

TABLE IV
Mapping of Code Quality Attributes of Dalla Palma et al.'s
paper [3] to ISO/IEC 25010:2011

ISO 25010:2011 Characteristics	Direct Mapping	Indirect Mapping	
Functional Suitability	Include,Ensure	Variables	
Performance Efficiency	Loops	Regex, Conditions, Decisions, Math Operation	
Compatibility	X	×	
Portability	Paths,Import Playbook,Import Role,Include Role,Include Task,Import Task,Include Vars	Blocks	
Reliability	Ensure,File Mode,File,Error Handling,Ignore Errors	Blocks	
Security	SSH Key,File Mode	Suspicious Comments, Deprecated Keywords, Deprecated Modules	
Usability	Comment	User Interactions	
Maintainability	Comment	Line Blank, Line Source Code, Blank Space Between Words, Avg Play Size, Avg Task Size, Unique Names, Length of Tasks, Entropy,Blocks, Variable Names	

No Mapping
Play, Roles, URL, Commands, Fact Modules, Filters,
Lookups, Keys, (External, Distinct) Modules

## Legend:

× - No IaC code attributes maps to ISO 25010:2011 characteristics

The IaC quality categories in our framework are Metadata, Code Structure, Code Sophistication, Code Maintainability, Code Security, Function and Purpose, Error Handling, Automation, and Code Integration. Fig. 1 shows these categories on the left hand side and also displays the relationship of each category with specific ISO/IEC 25010:2011 characteristics on the right hand side. For instance, Metadata relates to maintainability and reliability by providing critical information about versioning, dependencies, and configuration parameters within the infrastructure code, which are vital for understanding and managing the code effectively.

The categories in the framework have been developed to emphasize the importance of well-organized and modular code (Code Structure), the use of advanced syntax (Code Sophistication), and the ease of modifying

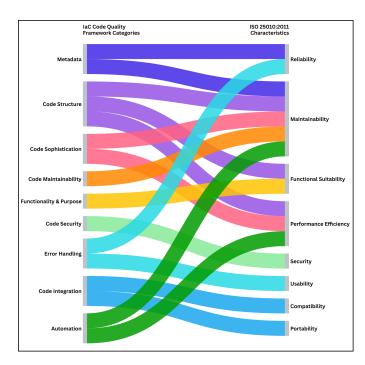


Fig. 1. Aligning of Our Proposed IaC Code Quality Framework to ISO/IEC 25010:2011 Characteristics

code to accommodate changes (Code Maintainability). It also highlights the importance of using security practices (Code Security), ensuring the code fulfills its intended functions (Function and Purpose), managing errors effectively (Error Handling), automating infrastructure processes to enhance efficiency (Automation), and enabling seamless integration across different environments (Code Integration).

Using the same methodology as Rahman et al.'s [2] and Dalla Palma et al. [3], we then mapped code quality attributes for each of the categories. To achieve this, data was collected from Ansible Galaxy repositories without factoring in popularity metrics like downloads or user ratings. This approach mitigated selection bias and allowed for an analysis that included a wide range of contributors. By incorporating metadata attributes not considered in earlier studies, such as those by Dalla Palma et al.[3] and Begoug et al.[4], the analysis identified 95 unique Ansible-specific code attributes, contributing to a total of 104 attributes. These attributes were subsequently mapped to the proposed IaC code quality categories, as presented in Table V.

# STEP 3: COMPARISON OF CURRENT RESEARCH MAPPING TO ISO/IEC 25010:2011 AND THE PROPOSED IaC CODE QUALITY FRAMEWORK

In this subsection, we map the code attributes identified by Rahman et al. [2] and Dalla Palma et al. [3] onto our proposed IaC Code Quality Framework to evaluate its applicability. These mappings can be seen in Table VI and Table VII. Rahman et al.'s attributes were successfully incorporated into our framework's categories, including previously unmapped attributes (see Table III & Table IV)

TABLE V
SUMMARY OF IAC CODE ATTRIBUTES MAPPED TO THE CODE QUALITY
FRAMEWORK CATEGORIES

IaC Code	IaC Code Attributes		
Quality	Tac Code Attributes		
Category			
Metadata	Download count, Tag, Total versions, Average update		
	time, Dependencies, Supported platform count, Stars,		
	Forks, Open issues, Minimum Ansible versions, version		
	release time		
Code	Count of lines in YAML/YML files, Lines of code in		
Structure	YAML/YML files, Task count, Template files, Count of		
	directories, Variables, Files count, Line blanks, Lines		
	in source code, Blank spaces between words, Average		
	play size, Average task size, Length of task, Unique names, Names with variables, Ensure		
Code			
Sophistication	Entropy, Loops, Conditions, Decisions, Mathematical		
•	operations, Regex, Lookups, Include, Keys, External modules, Distinct modules, Fact modules		
Code	modules, Distilict modules, Pact modules		
Maintainability	README count, README word count, License file		
	presence, Version information, Entropy, Comments, User interactions		
Functionality	README count, Fact modules, Comments		
& Purpose	READIVIE Count, Fact modules, Comments		
Code			
Security	SELinux, Firewalld, apt_key, passwd, Vault, stat, SSH		
	keys, File modes, Suspicious comments, Deprecated keywords, Deprecated modules		
Error	·		
Handling	Debug, failed_when, changed_when, Rescue,		
Ü	Always, Retry, Until loops, any_errors_fatal settings, max_fail_percentage, Delays, Error handling blocks,		
	ignore_errors, Blocks		
Automation	Become, vars, Handlers, Tags, check_mode,		
	environment, no_log, local_action, Fetch modules,		
	Paths, Commands, URLs, Plays, Roles, External		
G 1	modules, Distinct modules, Fact modules, Filters		
Code Integration	URI modules, URLs, wait_for modules, Rsync usage,		
integration	win_service, add_host, Git, Import playbooks, Import		
	roles, Import tasks, Include roles, Include tasks, Include		
	vars		

of "URL" and "Command," demonstrating the framework's ability to integrate attributes from prior research. Similarly, all 46 quality attributes from Dalla Palma et al. [3] for Ansible scripts can be mapped to our framework, illustrating its flexibility and applicability across different IaC tools and code quality assessments. This mapping process underscores the framework's potential as a unifying model for IaC code quality evaluation. Our IaC Code Quality Framework encompasses all attributes from both studies, aligning more closely with the unique aspects of IaC scripts and addressing elements not covered by general software quality models. This demonstrates that our framework is tailored to assessing IaC code quality.

➤ RQ3 Takeaway Message: Our IaC Code Quality Framework incorporates categories aligned with ISO/IEC 25010:2011 to encompass all identified IaC code attributes. It overcomes the limitations of the ISO standard in representing IaC-specific features, offering a holistic model for assessing IaC code quality.

TABLE VI
MAPPING RAHMAN ET AL.'S [2] STUDY TO OUR PROPOSED IAC CODE
QUALITY FRAMEWORK

IaC Code Quality Category	Direct Mapping	Indirect Mapping	No Mapping (Previously)
Metadata	×	×	×
Code Structure	File	Attribute, Lines of Code	×
Code Sophistication	Include,Ensure, Require	×	×
Code Maintainability	Comment	×	×
Functionality & Purpose	Comment	×	×
Code Security	SSH Key,File Mode	Hard-Coded String	×
Error Handling	×	×	×
Automation	×	×	URL, Commands
Code Integration	Include,Ensure, Require	×	×

#### Legend:

X - No IaC code attributes maps to our proposed IaC code quality category

#### IV. COMPUTATION OF IAC CODE ATTRIBUTES

In this section, we explain how we computed the code attributes for IaC. We began by counting the frequency of each code attribute and then adjusted these metrics per 100 lines of code, denoted by  $m_{100}$ , allowing comparisons between repositories of different sizes. Subsequently, we normalized the metrics to the range 0–1, denoted by  $(nm_{100})$ , which facilitates direct comparison by placing all metrics on a common scale, regardless of their original units or magnitudes.

Some tabulated metrics represent negative measures. For instance, higher counts for Suspicious Comments, Use of *passwd*, Deprecated keywords, and Deprecated modules indicate poorer code quality. For such metrics, their normalisation values were inverted using Equation 1. In doing so, this meant all the calculated metrics were in a form independent of repository size and aligned to the scales for consistent interpretation and aggregation.

$$nm_{100,neg} = 1 - \frac{m_{100,neg}}{\max(m_{100,neg})}$$
 (1)

Overall category scores per 100 lines of code were calculated by aggregating the normalized positive and negative metrics, weighted according to their importance, using the equation:

$$cs_{100} = \frac{\sum_{i=1}^{m} (nm_{100,i} \times w_i) + \sum_{j=1}^{n} (nm_{100,neg,j} \times w_j)}{N}$$
(2)

where  $nm_{100}$  and  $nm_{100,neg}$  are the normalized results of positive and negative metrics, respectively,  $w_i$  and  $w_j$  are the

TABLE VII
MAPPING DALLA PALMA ET AL.'S [3] STUDY TO OUR PROPOSED IAC
CODE QUALITY FRAMEWORK

IaC Code	Direct	Indirect	No Mapping
Quality Category	Mapping	Mapping	(Previously)
Metadata	X	X	X
Code Structure	File	Variables, Lines Blank, Lines Source Code, Blank Space Between Words, Avg Play Size, Avg Task Size, Length Tasks, Unique Names, Names with Variables	×
Code Sophistication	Include,Loops	Entropy, Conditions, Decisions, Math Operations, Regex	Lookups,Keys, (External, Fact, Distinct) Modules
Code Maintainability	Comment	Entropy,User Interaction	×
Functionality & Purpose	Comment	×	×
Code Security	SSH Key,File Mode	Suspicious Comments, Deprecated Keywords, Deprecated Modules	×
Error Handling	Error Handling Blocks,Ignore Errors	Blocks	×
Automation	Paths	×	Commands, URL, Filters, Play,Roles, (External, Distinct, Fact) Modules
Code Integration	Import Playbook,Import Role,Import Task,Include Role,Include Task,Include Vars	×	×

#### Legend:

X - No IaC code attributes maps to our proposed IaC code quality category

weights assigned to each positive and negative attribute, and N=m+n is the total number of positive (m) and negative (n) attributes incorporated in the calculation. The total score was derived by summing the category scores across all nine categories, resulting in a score ranging from 0–9, as each category score ranges from 0–1. This methodology ensured that each repository's code quality was assessed uniformly, accounting for size and the significance of each metric.

To measure the *Total Score*, the category scores per 100 lines of code are summed across all nine categories into a single, interpretable score using the following equation:

Total Score = 
$$\sum_{i=1}^{9} (cs_{100,i})$$
 , (3)

where  $cs_{100,i}$  represents the category score per 100 lines of code for each category in the proposed IaC framework, and *Total Score* denotes the aggregate score per 100 lines of code.

The assignment of weights determines the importance of attributes in code quality assessment. Weights ( $w_i \& w_j$  as shown in Equation 2) ranging from 0–1 allow for the prioritization of attributes, influencing the overall quality score. This flexibility allows organizations to:

- Measure code quality in accordance with software development practices and policies.
- Encode policies into quality measurement by assigning weights to attributes.

In our study, a weighting approach assigning equal importance to all attributes was employed to avoid bias in the evaluation. Utilizing a weighted average in the scoring model enables:

- Adjustments in response to changes in organizational policies.
- Maintenance of evaluation relevance as priorities evolve.

#### V. TEMPORAL DYNAMICS OF CODE QUALITY

• **RQ4:** How does the proposed IaC code quality framework perform when applied to real-world IaC repositories, and what insights can be gained?

In this section, we apply our framework on IaC scripts and present two primary observations: the time series analysis and an assessment of observable trends overtime of the quality of IaC and specifically Ansible scripts. This analysis uses data from 11,279 Ansible Galaxy repositories with complete records to evaluate code quality across various IaC practices within the Ansible community. The time series graph illustrates changes in mean scores over time, along with repository counts for each six-month period. Using a dual-axis layout, the graphs in Fig. 2 and Fig. 3 facilitate a direct comparison between repository quality (mean scores) and repository counts over time. Each bar is labeled with the repository count, allowing for an examination of changes in dataset composition. The mean scores for each interval reflects the number of repositories included, which varies according to the dataset composition for that period. Fig. 2 illustrates this for total score over all categories while Fig. 3 shows individual categories.

This longitudinal analysis identifies trends in repository quality and quantity, which may suggest changes in practices within the Ansible community. Using this methodology, we can examine how evolution influences mean scores over time, providing insights into broader IaC patterns within the Ansible

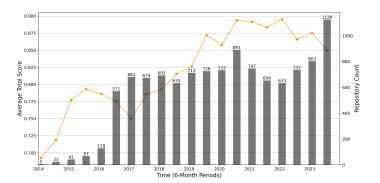


Fig. 2. 6-Month Averages of Total Score Across Time

Galaxy ecosystem. An initial review of the graphs reveals an upward trend in scores; however, further analysis indicates that the scores remain independent of repository count over time. One notable trend observed was an increase in average total score as shown in Fig. 2 during the period from mid-2020 to mid-2022, corresponding with the COVID-19 pandemic [28]. This period saw a global shift towards online services, leading many developers to prioritize IaC, with a focus on enhancing IT system resilience and security. This increase in repository quality during the pandemic years contrasts with data from the broader 2014–2023 period, highlighting a trend of increased code quality and attention to IaC standards in response to heightened demand for robust digital infrastructure.

Lastly, an Ordinary Least Squares (OLS) regression [29] was performed with six-month intervals as the independent variable and the mean scores as the dependent variable from the data shown in the graphs Fig. 2 and Fig. 3. Regression parameters, where slope values reveal trends over time as illustrated in Fig. 4.

Notably, the *Total Score* exhibited a consistent increasing trend which indicates a steady improvement in overall code quality over time. Similarly, the *Metadata Score* demonstrated the most significant increasing trend which reflects user's growing emphasis on the comprehensive metadata to enhance repository usability. Increasing trends were also identified in the *Structure Score* and *Security Score*, suggesting improvements in organizational structure and increased attention to security practices to align with best practices in the IaC industry. Additionally, the *Error Handling Score* showed a significant increase which indicates a growing focus on error management and self-healing capabilities.

Conversely, certain categories exhibited declining trends. The *Sophistication Score* showed a decreasing trend, suggesting a decrease in the use of diverse keywords in repositories. The *Functionality & Purpose Score* also demonstrated a statistically significant decreasing trend, indicating a potential shift towards simpler and more focused repositories. Similarly, the *Automation Score* decreased over time, indicating a decline in the usage of automation-specific keywords. This is likely due to the improvements in the base Ansible-core, which gradually incorporates more advanced

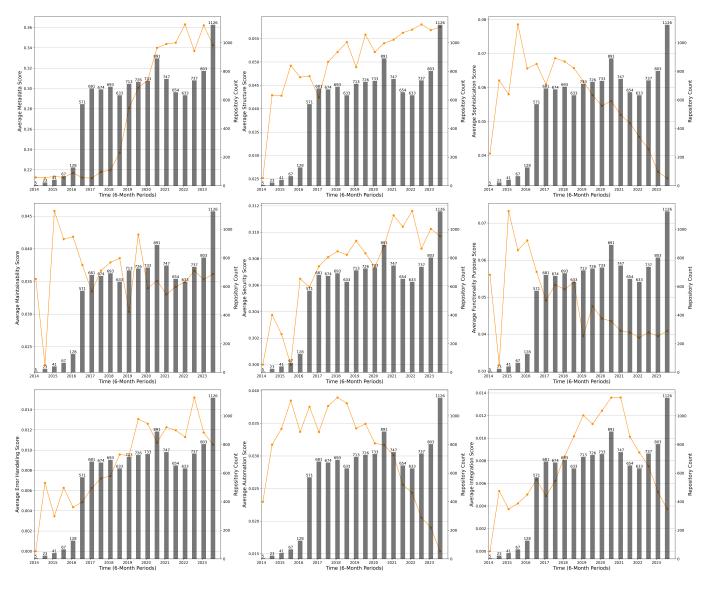


Fig. 3. 6-Month Average of Category Scores Across Time

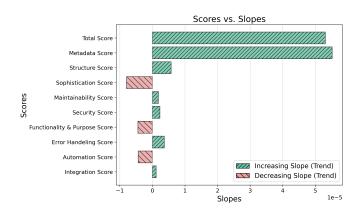


Fig. 4. Observable Trends

keywords from various third-party modules.

Overall, these trends highlight a community-wide focus on improving code quality in areas such as metadata completeness, structural organization, security, and error handling. The decline in usage of attributes related to sophistication, automation, and functionality may reflect a shift towards simplicity and maintainability, indicating evolving priorities within the community towards more streamlined IaC practices.

➤ RQ4 Takeaway Message: Using our IaC code quality framework to analyze real-world Ansible repositories demonstrated overall improvements in code quality and offered insights into evolving practices within the ansible community.

#### VI. DISCUSSION

• RQ5: In what ways can the proposed IaC code quality framework enhance the assessment of IaC scripts in practice?

The proposed IaC code quality framework evaluates Ansible scripts by establishing definitions backed by standardization, ensuring consistent assessments across repositories and authors. Practitioners can benchmark their code against industry standards using quantifiable metrics that identify strengths and weaknesses. The framework emphasizes the importance of detailed metadata for enhancing repository discoverability and usability, providing insights for tracking progress and adjusting strategies. For researchers, the framework provides a basis for comparative studies across IaC tools, the incorporation of additional attributes and metrics, and the utilization of large-scale repository data for quantitative analysis of IaC development practices. Researchers can examine relations between specific metrics and overall quality or explore deeper connections using various data analysis techniques.

#### A. LIMITATIONS AND RECOMMENDATIONS

This study identifies key limitations and proposes recommendations for future research.

#### **■** Limitations:

# • Metadata Completeness Criterion

- The analysis included 11,279 repositories out of approximately 26,900 in Ansible Galaxy (42%) due to the requirement for complete metadata.
- This selection criterion exclude repositories with incomplete metadata, potentially introducing bias and omitting repositories with distinct characteristics.

# • Tool-Specific Scope

 The study focused exclusively on Ansible repositories from Ansible Galaxy, which limits the applicability of findings to other IaC tools.

## • Framework Weighting Influence

 The assigned weights in the framework metrics can affect the interpretation of their importance, necessitating thorough testing and careful evaluation to mitigate bias.

# • Correlations Analysis

 While trends were identified, our study needs to establish clear correlations between specific code attributes and category-specific quality scores.

## **■** Recommendations:

- Expand the scope to include other IaC tools and repository hosting platforms.
- Refine the relationship between organizational policies and assigned weights based on empirical evidence or expert feedback.
- Incorporate qualitative research methods, such as:

- Conducting surveys or interviews with developers to complement quantitative analyses.
- Gaining insights into coding practices and community norms.
- Adjust metric definitions and experiment with weightings to align with current best practices.
- Identifying and categorizing outliers, whether beneficial or detrimental, remains a critical aspect of data analysis that warrants further investigation.

➤ RQ5 Takeaway Message: Our study provides metrics for practitioners to assess script quality and offers researchers a basis for deeper exploration of IaC development practices.

#### VII. CONCLUSION

This paper has focused on the growing trend of IaC for automated configuration through scripting, highlighting the lack of standardized code quality evaluation aligned with a software quality standard such as ISO/IEC 25010:2011. To bridge this gap, we have proposed an IaC quality framework for evaluating Ansible scripts using definitions aligned with standards for consistent assessments across repositories. In this process, we identified 95 unique Ansible-specific code attributes. With it practitioners, for instance, can benchmark their code against industry standards using quantifiable metrics to identify strengths and weaknesses. Emphasizing metadata importance for repository usability, we have formulated nine IaC specific code quality categories. Taking a decade's worth of Anisble-based repositories, analysis reveals a trend towards enhancing code quality in areas like metadata completeness, structural organization, security, and error handling. A decreased emphasis on sophistication, automation, and functionality was also noted, which suggests a shift towards simplicity of code, indicating evolving priorities toward more streamlined IaC practices.

#### VIII. DATA AVAILABILITY

The data supporting the findings of this study are openly and anonymously available on Figshare at: https://figshare.com/s/b78d2c82767ab7d86beb

#### ACKNOWLEDGMENT

We appreciate the valuable feedback provided by the anonymous reviewers. The authors would also like to acknowledge funding support from the New Zealand Ministry of Business, Innovation, and Employment (MBIE) for project UOWX1911, "Artificial Intelligence for Human-Centric Security".

#### REFERENCES

- [1] International Organization for Standardization, "Iso / iec 25010: 2011 systems and software engineering systems and software quality requirements and evaluation (square) system and software quality models," 2013. [Online]. Available: https://www.iso.org/obp/ui/#iso:std: iso-iec:25010:ed-1:v1:en
- [2] A. Rahman and L. Williams, "Source code properties of defective infrastructure as code scripts," *Information and Software Technology*, vol. 112, p. 148–163, Aug. 2019. [Online]. Available: http://dx.doi.org/ 10.1016/j.infsof.2019.04.013
- [3] S. D. Palma, D. D. Nucci, F. Palomba, and D. A. Tamburri, "Towards a catalogue of software quality metrics for infrastructure code," ArXiv, vol. abs/2005.13474, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:218900954
- [4] M. Begoug, M. Chouchen, and A. Ouni, "TerraMetrics: An open source tool for Infrastructure-as-Code (IaC) quality metrics in Terraform," in *Proceedings of the 32nd IEEE/ACM International Conference on Program Comprehension*, ser. ICPC '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 450–454. [Online]. Available: https://doi.org/10.1145/3643916.3644439
- [5] Ansible Community, "Ansible Galaxy," Red Hat, Inc., 2023. [Online]. Available: https://galaxy.ansible.com/
- [6] R. Wang, Infrastructure as Code, Patterns and Practices: With examples in Python and Terraform, ser. ITpro collection. Manning, 2022.
- [7] Amazon Web Services, Inc., "AWS CloudFormation," 2023. [Online]. Available: https://aws.amazon.com/cloudformation/
- [8] J. Boero, HashiCorp Packer in Production: Efficiently Manage Sets of Images for Your Digital Transformation Or Cloud Adoption Journey. Birmingham, UK: Packt Publishing, 2023.
- [9] S. P. Kane and K. Matthias, *Docker: Up and Running*, 3rd ed. Sebastopol, CA: O'Reilly Media, 2023.
- [10] S. Winkler, Terraform in Action. Shelter Island: Manning, 2021.
- [11] Terraform, "Terraform registry," 2024. [Online]. Available: https://registry.terraform.io/
- [12] Azure, "Azure Resource Manager (ARM) templates," 2024. [Online]. Available: https://learn.microsoft.com/en-us/azure/ azure-resource-manager/templates/
- [13] ——, "Azure quickstart templates," 2024. [Online]. Available: https://github.com/Azure/azure-quickstart-templates
- [14] Google Cloud, "Google cloud deployment manager," 2024. [Online]. Available: https://cloud.google.com/deployment-manager
- [15] B. Meijer, L. Hochstein, and R. Moser, Ansible: Up and Running, 3rd ed. Sebastopol, CA: O'Reilly Media, 2022.

- [16] D. Sandilands, Puppet 8 for DevOpsEngineers: Automate Your Infrastructure at an Enterprise Scale. Birmingham, UK: Packt Publishing, 2023.
- [17] Puppet, "Puppet forge," 2024. [Online]. Available: https://forge.puppet. com/
- [18] E. Waud, Mastering Chef Provisioning. Birmingham, UK: Packt Publishing, 2016.
- [19] SaltStack, "SaltStack (Salt)," 2024. [Online]. Available: https://saltproject.io/
- [20] —, "SaltStack formula (Salt)," 2024. [Online]. Available: https://github.com/saltstack-formulas
- [21] A. Cunin, "'yamllint' documentation: A linter for YAML files," 2023. [Online]. Available: https://yamllint.readthedocs.io
- [22] W. Thames and contributors, "ansible-lint checks playbooks for practices and behavior that could potentially be improved," 2023. [Online]. Available: https://ansible-lint.readthedocs.io
- [23] J. Schwarz, A. Steffens, and H. Lichter, "Code smells in infrastructure as code," in 2018 11th International Conference on the Quality of Information and Communications Technology (QUATIC), 2018, pp. 220–228.
- [24] P. R. Reddy Konala, V. Kumar, and D. Bainbridge, "SoK: Static configuration analysis in infrastructure as code scripts," in 2023 IEEE International Conference on Cyber Security and Resilience (CSR), 2023, pp. 281–288.
- [25] International Organization for Standardization, "ISO/IEC 9126-1: Software Engineering - Product Quality - Part 1: Quality Model," 2001. [Online]. Available: https://www.iso.org/standard/22749.html
- [26] —, "Iso/iec 25010:2023 systems and software engineering systems and software quality requirements and evaluation (square) system and software quality models," December 2023. [Online]. Available: https://www.iso.org/obp/ui/en/#iso:std:iso-iec:25010:ed-2:v1:en
- [27] Ansible Community, "Ansible," Red Hat, Inc., 2023. [Online]. Available: https://www.ansible.com/
- [28] A. Feldmann, O. Gasser, F. Lichtblau, E. Pujol, I. Poese, C. Dietzel, D. Wagner, M. Wichtlhuber, J. Tapiador, N. Vallina-Rodriguez, O. Hohlfeld, and G. Smaragdakis, "The lockdown effect: Implications of the COVID-19 pandemic on Internet traffic," in *Proceedings of the ACM Internet Measurement Conference*, ser. IMC '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 1–18. [Online]. Available: https://doi.org/10.1145/3419394.3423658
- [29] J. H. Stock and M. W. Watson, *Introduction to Econometrics*, 4th ed. Boston, MA: Pearson, 2020.