RESEARCH-ARTICLE

TerraMetrics: An Open Source Tool for Infrastructure-as-Code (IaC) Quality Metrics in Terraform

MAHI BEGOUG, School of Higher Technology, Montreal, QC, Canada

MOATAZ CHOUCHEN, School of Higher Technology, Montreal, QC, Canada

ALI OUNI, School of Higher Technology, Montreal, QC, Canada

Open Access Support provided by:

School of Higher Technology

Published: 15 April 2024

Citation in BibTeX format

ICPC '24: 32nd IEEE/ACM International Conference on Program Comprehension
April 15 - 16, 2024
Lisbon, Portugal

Conference Sponsors:
SIGSOFT

https://doi.org/10.1145/3643916.3644439
ISBN: 9798400705861

TerraMetrics: An Open Source Tool for Infrastructure-as-Code (IaC) Quality Metrics in Terraform

Mahi Begoug
ETS Montreal, University of Quebec
Montreal, QC, Canada
mahi.begoug.1@ens.etsmtl.ca

Moataz Chouchen
ETS Montreal, University of Quebec
Montreal, QC, Canada
moataz.chouchen.1@ens.etsmtl.ca

Ali Ouni
ETS Montreal, University of Quebec
Montreal, QC, Canada
ali.ouni@etsmtl.ca

ABSTRACT

Infrastructure-as-Code (IaC) constitutes a pivotal DevOps methodology, leading edge of software deployment onto cloud platforms. IaC relies on source code files rather than manual configuration to manage the infrastructure of a software system. Terraform, an IaC tool and its declarative configuration language named HCL, has recently garnered considerable attention among IaC practitioners. Like other software artefacts, Terraform files could be affected by misconfigurations, faults, and smells. Therefore, DevOps practitioners might benefit from a quality assurance tool to help them perform quality assurance activities on Terrafrom artefacts. This paper introduces TerraMetrics, an open-source tool designed to characterize the quality of Terraform artefacts by providing a catalogue of 40 quality metrics. TerraMetrics leverages the Terraform Abstract Syntax Tree (AST) to extract the metric list, offering a potentially enduring solution compared to conventional regular expressions. This tool comprises three main components: (i) a parser transforming HCL code into an AST, (ii) visitors that traverse the AST nodes to extract the metrics, and (iii) collectors for storing the collected metrics in JSON format. The TerraMetrics tool is publicly available as an Open Source tool, with a demo video, at: https://github.com/stilab-ets/terametrics.

KEYWORDS

Infrastructure-as-Code, Terraform, HCL, quality metrics, AST

ACM Reference Format:
Mahi Begoug, Moataz Chouchen, and Ali Ouni. 2024. TerraMetrics: An Open Source Tool for Infrastructure-as-Code (IaC) Quality Metrics in Terraform. In *32nd IEEE/ACM International Conference on Program Comprehension (ICPC '24), April 15–16, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3643916.3644439

1 INTRODUCTION

Infrastructure-as-Code (IaC) is a DevOps practice designed to streamline the configuration of software systems infrastructure. It employs source code files for configuring infrastructure, replacing the manual setup or human involvement in traditional software systems. Several tools, such as Ansible [21], Terraform [20], and Puppet [30], facilitate the implementation of IaC. Recently, Terraform has gained popularity among IaC practitioners [8]. Terraform facilitates the provisioning of software system infrastructure through its configuration language called HCL [6]. HCL language is declarative, enabling practitioners to define intricate infrastructures. HCL relies on block units similar to methods or functions in object-oriented programming where developers may encapsulate specific behaviour and business rules. These blocks are the main bricks to define and manage the software systems infrastructure [13, 29].

Despite its popularity, practitioners face several challenges with Terraform, such as misconfigurations, technical debt, and security vulnerabilities in its configuration files, which can impede practical adoption and lead to outages of infrastructure [12]. For example, a recent issue arose in Terraform files used to manage the infrastructure of `GitLab.com` [38], impacting all users, making the platform temporarily unavailable. The root causes of this issue were related to the outdated configuration of the production database configuration, including erroneous blocks. Although there are Terraform-related developer support tools like `checkov` [16], `tfsec` [9], and `terrascan` [33], these tools primarily focus on detecting security issues and vulnerabilities without providing a characterization regarding the quality of the block (*e.g.,* block complexity, references, and size) that could be used to assess the maintainability of the HCL code. Furthermore, to illustrate the current state of IaC tools, Konala et al. [23] have studied the existing quality assurance of IaC scripts. They found that Ansible and Puppet are the most commonly supported, however, they only showed `tfsec` for Terraform-based scripts. We believe that Terraform practitioners still need tool support to assess and improve the quality of their code and its maintainability.

In this paper, we aim to provide a comprehensive collection of quality metrics that characterize and quantify the features of HCL code used in Terraform scripts. To this end, we implemented `TerraMetrics`, a tool to automatically measure 40 metrics at the block level within the Terraform configuration file. Depending on the users' case, these metrics can be aggregated to the file or package level. Our tool is based on static code analysis, an off-line technique that allows the analysis without execution, saving resource consumption. Initially, our tool parses the HCL code provided as input to build its Abstract Syntax Tree (AST) structure. After that, we iterate through the AST tree and analyze its nodes. Then, we calculate the HCL code metrics depending on the node types. Finally, we save the identified metrics in a readable JSON format.

To evaluate the usefulness of `TerraMetrics`, we performed an experimental study on 3 widely used Terraform modules designed for different cloud platforms (*i.e.,* AWS [1], GCP [3], and Azure

ICPC '24, April 15–16, 2024, Lisbon, Portugal

ACM ISBN 979-8-4007-0586-1/24/04…$15.00
https://doi.org/10.1145/3643916.3644439

[2]. The tool has shown a prominent performance in terms of understandability with an average precision of 88% and a recall of 97%.

Open source tool and documentation. We provide our TerraMetrics implementation as an Open Source tool available on Github [7], along with documentation and demonstration video.

2 BACKGROUND & RELATED WORKS

Recently, HCL has gained attention among IaC practitioners, as highlighted in the GitHub 2023 yearly report [19]. In this section, we present an overview of the HCL language and summarize related work on IaC tools.

2.1 HashiCorp Configuration Language (HCL)

HCL, a domain-specific language (DSL), has its interpreter built in GoLang [10, 29]. HCL enriches the conventional declarative code paradigm by incorporating features such as loops and conditionals. These extensions take advantage of the capabilities of imperative languages to define configurations that may not be suitable for a standard declarative approach. Conceptually, an HCL configuration can be seen as a compilation of attribute-value pairs declared within *blocks*. The attribute values encompass literals, conditionals, and loop expressions. Moreover, they can take the form of template expressions, which facilitates dynamic loading of strings. When integrated into Terraform files, HCL attributes could be contained in various block types: variable, module, local, resource, data, terraform, provider, and output [10, 15, 26, 27].

Figure 1 illustrates an example of an HCL code executable by Terraform that aims to create a Simple Storage Service (S3) on the Amazon platform. The keyword `resource` denotes a cloud resource or component, while `aws_s3_bucket` specifies the resource type. The `s3_service` keyword is a unique identifier for the declared resource. Moreover, the curly braces encapsulate the block content (*i.e.,* body), where attributes like 'bucket' are declared. At the same time, this block contains nested blocks that can be used for a more detailed resource configuration.

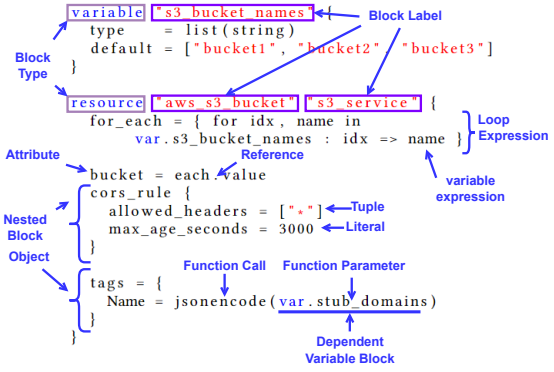

Figure 1: An example of HCL code snippet in Terraform.

2.2 Related Works

This subsection summarizes existing works for static analysis of IaC. Sharma et al. [37] proposed a catalogue of 24 code smells that could be present in Puppet scripts, categorized into implementation and design smells. They implemented the code smells catalogue in the Puppeteer tool [36]. Dalla et al. [17] then proposed a catalogue of 46 metrics to improve the quality of Ansible scripts and implement them in an open-source package named AnsibleMetrics [18]. Meanwhile, Borovits et al. [14] implemented DeepIaC, a tool for identifying antipatterns in Ansible scripts based on deep learning. Later, Rahman et al. [32] delved into security smells in Puppet scripts and introduced SLIC, a tool for identifying seven security smells. Based on Rahman et al. [32] security metrics, Saavedra et al. [34] introduced GLITCH, an agnostic security smell detection tool in different IaC technologies such as Chef, Puppet, and Ansible. Later, Opendeeck et al. [28] proposed GASEL, a security smell detector customized for the Ansible IaC language.

The IaC research effort's primary emphasis was on platforms like Puppet and Ansible. Moreover, current IaC tools focus primarily on identifying security issues that can cause infrastructure outages. Consequently, there is no support for quality assurance tools for less explored IaC platforms, including Terraform, an emerging and widely adopted IaC provisioning tool. Thus, we aim to develop a Terraform quality assurance framework that can be used to identify Terraform code issues such as defects, code smells, and security issues and help them in code review activities.

3 TERRAMETRICS ARCHITECTURE

TerraMetrics is implemented as an Open Source command line-based tool, available as a standalone `Jar` file. Figure 2 shows a high-level overview of the TerraMetrics architecture. TerraMetrics has four modules: (1) the HCL parser module, (2) AST visitors, (3) the Metrics calculator, and (4) the Metrics collector. In the following, we provide details for each module, as well as the usage guide.

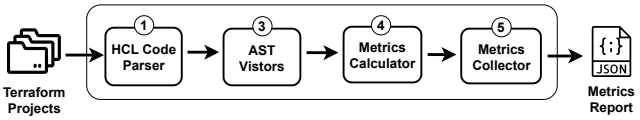

Figure 2: High-level architecture of TerraMetrics

HCL Code Parser. Initially, this module identifies Terraform files, specifically those concluding with the ".tf" extension. The tool utilizes the HCL parser from the SonarQube Terraform parser [39] to build the Abstract Syntax Tree (AST) for each identified ".tf" file. The SonarQube Terraform Package is employed for two primary reasons. First, it helps to exclude HCL files with syntax errors. Second, it enables accurate metric measurement in intricate cases where nested configurations (*e.g.,* blocks under blocks) may exist. In the AST structure of the HCL code, the Terraform file serves as the root node, which includes various nodes (*i.e.,* non-binary structure). Each node at the initial level represents the blocks of the file, which may also function as containers for attributes considered sub-nodes. This methodical transformation ensures a comprehensive representation of the HCL code structure.

AST Visitors. We employ visitors pattern [22] to traverse various types of tree nodes in the AST. We define a visitor with an appropriate visit method for each tree node type. For example, when identifying the loop expression (*i.e.,* for), we instantiate a `LoopsVisitor` to discern all expressions contained in a given block recursively. Subsequently, we check whether each visited expression's type corresponds to a `ForTreeImpl` defined by the HCL

parser. In our implementation, we intentionally decouple the visitors from the parser module to reach loose decoupling between the classes, earn the single responsibility principle, and for further extensions.

Metrics Calculator. This module is an essential component in the TerraMetrics architecture, which measures the HCL code metrics. Our metric calculation operates at the block level. This level constitutes building bricks of the Terraform artefacts. It enables practitioners to formulate their infrastructure configurations modularly, like traditional development in which developers use classes and methods to abstract a specific business logic. In this module, the classes receive data from visitors who have traversed the previously parsed HCL code. These classes measure the metrics across distinct scopes, containing the total, average, minimum, and maximum numbers. We draw inspiration from the Understand tool [35], which is used for the analysis of traditional languages (*e.g.,* Java, PHP) and offers features with different scopes. This technique offers practitioners and researchers a more comprehensive analysis of HCL at the block level, easily aggregated at the file or package level.

Metrics Collector. This module takes the calculated metrics and saves them in JavaScript Object Notation (JSON) format. We selected this format to facilitate users in importing data into various systems and platforms. Within this component, the decorator design pattern is used. This pattern enables the appending of new measured metrics to collector classes at runtime without disrupting the existing code responsible for metric measurement [22].

4 TERRAMETRICS USAGE SCENARIO

TerraMetrics is distributed as a `Jar` file. Its provision as a standalone executable, as opposed to a plugin, eliminates users' need to have a particular Integrated Development Environment (IDE) installed on their operating system for quantifying HCL metrics. Similar to the other tools (*e.g.,* tsDetect [31], FindBugs [4], and PMD [5]), TerraMetrics is operational through the command line, facilitating seamless integration with deployment pipelines.

TerraMetrics takes as input Terraform files with the extension ".tf", which can be a single file, a local folder, or even a GitHub repository. Our tool can be executed from the command line `terrametrics.jar` on any operating system, virtual machine or Docker container, provided that the `Java` system is installed, with a minimum version of 11. The command line can be executed by running the following command with specific parameters:

```
java -jar terrametrics.jar <parameters>
```

where the different parameters can used as follow:

```
-f, –file <File path to be analyzed>
-t, –target <File path for analysis output>
–local <Local directory path to be analyzed>
-r, –repo <GitHub repository URL to be analyzed>
-h, –help <?>
```

After execution, a JSON file containing all the metrics values will be created on the local directory path as provided by the user.

5 EVALUATION

We conducted a set of experiments to evaluate the correctness and usefulness of TerraMetrics.

Correctness Evaluation. To assess the correctness of TerraMetrics, we used a Test-Driven Development (TDD) [11] approach, which is an agile development technique that stresses the construction of any software system under testing. (1) We started by writing a failing test that tests the values of the metrics; Then, (2) we wrote a minimum code to make the test cases pass for each metric implementation. In this phase, the tested code was extracted from the official Terraform documentation. To evaluate the correctness of the metrics values, we randomly selected HCL source code fragments from `terraform-aws-modules/terraform-aws-eks` (see `base.tf` file in the TerraMetrics package test module). We reiterated to test the implementation and re-apply refactoring techniques (*i.e.,* extract method). By following the TDD approach in our tool development, we implement a cyclical process to get fast feedback, mitigate erroneous measures, and easily add new requirements and metrics. Thereafter, we selected three different open-source and widely used Terraform projects from GitHub, as detailed in Table 2. Each project originates from different organizations and is used on a specific cloud platform (*i.e.,* AWS, GCP, Azure). We then manually verified a representative random sample of blocs (confidence level = 95%) by manually counting the metrics values as compared to the values returned by TerraMetrics, We found that their TerraMetrics has an accuracy of 100%.

Usefulness Evaluation. We evaluated the usefulness of TerraMetrics to evaluate the readability of the code. Although there could be different scenarios to evaluate the usefulness of our tool, we focused on code understandability, one of the widely recognized assessments for software maintainability. We used the Cyclomatic Complexity metric as a proxy of understandability following the recent work of Lavazza et al. [24]. Thereafter, we run TerraMetrics on all Terraform files from our three projects. We then split the files into 3 categories based on the distribution of the Cyclomatic Complexity metrics values obtained by TerraMetrics : (1) "*Understandable*" (files with complexity values ranging from 1 to 23, *i.e.,* less than the first Quartile), (2) "*Somewhat understandable*" (files with complexity values ranging from 23 to 53, *i.e.,* between the first and third Quartile), and (3) "*Hard to understand*" (files with complexity values higher than 54, *i.e.,* higher than the third Quartile), as shown in Table 3. To assess the code understandability, two authors selected a random sample of 100 Terraform files from the three studied projects. The authors then conducted an independent manual inspection to manually label the inspected files to indicate whether they are: "Hard to understand", "Somewhat understandable", or "Understandable". We measured the Cohen's K [40] inter-rater agreement which yielded an initial agreement rate of 0.571 (moderate agreement). The authors resolved the disagreements that led to Cohen's K of 0.86 (perfect agreement). Later, we compared the manually labeled results and the identified intervals based on the complexity values identified by TerraMetrics using the average precision, recall, and F1 scores. We observe from Table 3 that the *Understandable* and *Hard to understand* align with complexity metrics with a high F1 score of 0.9, whereas the *Somewhat understandable* category shows less matching with manual labeling with an F1 score of 0.78. It is worth noting that we do not have a perfect F1 score for the Somewhat understandable class. Thus, using cyclomatic complexity as a proxy for code understandability is not always perfect, as pointed out in previous research [24, 25].

Table 1: The list of quality metrics implemented by TerraMetrics to identify the properties of the HCL source code.

Metric Name	Description	Aggregation
Comparison operators	Number of the occurrences of `==, !=, <, >, <=, >=` in AST nodes	Total, Avg, Max
Logical operators	Number of the occurrences of `!, &&, ||` within the nodes of a given block	Total, Avg, Max
Math operations	Number of `-, +, *, /, %` in tree nodes occurrences	Total, Avg, Max
Conditions	Number of conditional nodes	Total, Avg, Max
Loops	Number of occurrences of `for` nodes in a given block	Total, Avg, Max
Cyclomatic complexity	Number of independent paths (*i.e.,* number of loops + number of conditions + 1)	Total, Avg, Max
Function call	Number of function invocation nodes in a given block	Total, Avg, Max
Lookup function call	Number of `lookup` names in function invocation nodes	Total
Debugging functions Call	Number of `try, can, type` directive in function invocation nodes	Total
Deprecated functions	Number of `map, list` names in function invocation nodes	Total
Function parameters	Number of the children nodes of a function call node	Total, Avg, Max
Splat expressions	Number of splat expression (*i.e.,* similar to lambda expressions in Java) nodes that could optimize the loop expressions	Total, Avg, Max
References	Number of reference nodes that represent the access to variables	Total, Avg, Max
Objects	Number of object nodes	Total, Avg, Max
Element of objects	Number of the children nodes of an object node	Total, Avg, Max
Tuples	Number of tuple (*i.e.,* similar to an Array in Java) nodes	Total, Avg, Max
Element of tuples	Number of the children nodes of a tuple nodes	Total, Avg, Max
Index access	Number of index access nodes that represent the tuple elements access (*e.g.,* `exampleTuple[0]`)	Total, Avg, Max
Variables	Number of variables nodes that represent the variable expressions	Total, Avg, Max
Template	Number of template nodes that represent the template expressions to load dynamically string values	Total, Avg
Heredoc	Number of heredoc nodes that represent the multiline string expression expressed as `<<EOF ... EOF`	Total, Avg
Size of heredoc	Number of the lines contained in heredoc expressions	Total, Avg, Max
Attributes	Number of node attribute that represent the number block attributes	Total
Meta arguments	Number of meta arguments nodes that are special attributes used with resource blocks and modules to manage their functionality	Total
Line of code	Number of executed code lines	Total
Non-line of code	Number of empty lines and commented lines	Total
Depth of block	Size of a given block by using the start-end indexes of the nodes	Total
Dynamic blocks	Number of `dynamic` references of blocks	Total
Nested blocks	Number of nested block nodes within a block	Total
Depth Nested Blocks	The size of the nested blocks	Avg, Max, Min
Text Entropy	The quantity of information per block expressed as $-\sum_{t \in \text{tokens(block)}} p(t) \log_2(p(t))$	Total
Tokens	Number of tokens which are words separated by blank	Total, Avg, Max, Min
Literal	Number of literal nodes that represent a practically constant value (*i.e.,* string, boolean, or null)	Total
String values	Number of string nodes within literal nodes that represent the string values	Total
Star string	Number of "*" string node that could be as attributes values	Total
Wild card suffix string	Number of `":*"` string node that could be as attributes values	Total
Empty string	Number of the occurrences of "" string node that could be as attributes values	Total
Dependent Block	Number of the direct call dependency from a given block to another	Tool
Block type	Check if the type of a given block	Tool
Documentation field	Check if a given block that contains `description` attribute	Tool

Table 2: Statistics of three studied projects

Project	Creation date	LOC	# of Releases	# of Contributors	# of Stars
terraform-aws-modules/terraform-aws-eks	Jun 6, 2018	15,878	198	331	3,900
terraform-google-modules/terraform-google-kubernetes-engine	Aug 14, 2018	65,613	74	236	1,000
Azure/caf-terraform-landingzones	Dec 12, 2019	31,196	45	60	719

While the TerraMetrics tool can be helpful for practitioners interested in evaluating infrastructure code understandability, multiple other usage scenarios can be considered (cf. Section 6).

Table 3: Results for code complexity and understandability.

Understandability Category	Cyclomatic Complexity	Precision	Recall	F1
Understandable	1-23	1.0	0.91	0.95
Somewhat understandable	24-53	0.65	1.0	0.78
Hard to understand	54+	1.0	1.0	1.0

6 TERRAMETRICS APPLICABILITY

Practitioners. Practitioners can integrate TerraMetrics into their deployment workflow and provisioning toolkit. They can incorporate TerraMetrics into their code review process to improve code quality and foster a deeper understanding of the HCL code written by other maintainers. As an illustration, practitioners can use it to verify whether variable blocks are documented, including a description field that clarifies the role of each variable. TerraMetrics may also be invaluable in assisting practitioners during code refactoring. For example, metrics such as block depth, the number of nested blocks, and the number of dependent blocks can help practitioners evaluate the readability and maintainability of blocks.

Researchers. TerraMetrics, with its comprehensive set of metrics, serves as a valuable resource for researchers conducting empirical studies on software maintenance and quality. Researchers can explore correlations within Terraform configurations that may indicate code smells, diminishing the overall quality of the configuration.

Educators. The HCL is inherently declarative, designed to facilitate the definition of intricate infrastructure components. This nature may challenge beginner Terraform practitioners, as it diverges from traditional coding and imperative styles. Therefore, DevOps and software deployment educators can use TerraMetrics as a valuable resource to guide new practitioners and students. The metrics integrated into TerraMetrics provide a practical introduction to grasping the HCL language, covering its fundamentals such as types, block structures, expressions, and loops. We also provide in our replication package a meta-model as a class diagram that describes the HCL language, which could help educators and their students.

7 CONCLUSION AND FUTURE WORKS

This paper introduces TerraMetrics, a tool designed to quantify 40 HCL-related metrics automatically within Terraform files used to provision the infrastructure of the software system. The tool helps practitioners asses the quality of their Terraform artefacts and supports researchers conducting empirical studies in Iac. The paper clarified the architecture of TerraMetrics and outlined the 40 HCL quality metrics and how to compute them. Further, the usefulness of TerraMetrics is assessed through an empirical experiment on three widely used open-source projects, showing prominent performance with an average precision of 88% and a recall of 97% in terms of readability and understandability.

For future developments, we plan to enhance the visualization aspect of TerraMetrics by including a comprehensive user interface to facilitate metric analysis for practitioners. Additionally, we intend to integrate TerraMetrics as a plugin for popular Integrated Development Environments such as VsCode. Finally, we encourage the IaC community to download TerraMetrics and contribute to its extension.

REFERENCES

[1] [n. d.]. Amazon Web Services. https://aws.amazon.com/. Accessed on: Nov 1, 2023.
[2] [n. d.]. Azure. https://azure.microsoft.com/en-ca. Accessed on: Nov 1, 2023.
[3] [n. d.]. Cloud Google Platform. https://cloud.google.com/. Accessed on: Nov 1, 2023.
[4] [n. d.]. Find Bugs in Java Programs. https://findbugs.sourceforge.net/. Accessed on: Nov 1, 2023.
[5] [n. d.]. PMD Source Code Analyzer. https://pmd.github.io/. Accessed on: Nov 1, 2023.
[6] [n. d.]. Terraform Language Documentation. https://developer.hashicorp.com/terraform/language. Accessed on: Nov 1, 2023.
[7] [n. d.]. TerraMetrics: An Open Source Tool for Infrastructure-as-Code (IaC) Quality Metrics in Terraform. https://github.com/stilab-ets/terametrics. Accessed on: Nov 1, 2023.
[8] [n. d.]. The top programming languages. https://octoverse.github.com/2022/top-programming-languages. Accessed on: Nov 1, 2023.
[9] aquasecurity. [n. d.]. https://github.com/aquasecurity/tfsec. Accessed on: Nov 1, 2023.
[10] Martin Atkins. [n. d.]. HCL Syntax-Agnostic Information Model. https://github.com/hashicorp/hcl/blob/main/spec.md. Accessed on: Nov 1, 2023.
[11] Beck. 2002. *Test Driven Development: By Example*. Addison-Wesley Longman Publishing Co., Inc., USA.
[12] Mahi Begoug, Narjes Bessghaier, Ali Ouni, Eman Alomar, and Mohamed Wiem Mkaouer. 2023. What Do Infrastructure-as-Code Practitioners Discuss: An Empirical Study on Stack Overflow. In *17th International Conference on Empirical Software Engineering and Measurement (ESEM)*. 1–12 pages.
[13] Mahi Begoug, Moataz Chouchen, Ali Ouni, Eman Alomar, and Mohamed Wiem Mkaouer. 2024. Fine-Grained Just-In-Time Defect Prediction at the Block Level in Infrastructure-as-Code (IaC). In *ACM International Conference on Mining Software Repositories (MSR)*. 1–12 pages.
[14] Nemania Borovits, Indika Kumara, Parvathy Krishnan, Stefano Dalla Palma, Dario Di Nucci, Fabio Palomba, Damian A Tamburri, and Willem-Jan van den Heuvel. 2020. DeepIaC: deep learning-based linguistic anti-pattern detection in IaC. In *Proceedings of the 4th ACM SIGSOFT International Workshop on Machine-Learning Techniques for Software-Quality Evaluation*. 7–12.
[15] Yevgeniy Brikman. 2022. *Terraform: Up and Running*. " O'Reilly Media, Inc.".
[16] Checkov. [n. d.]. https://github.com/bridgecrewio/checkov. Accessed on: Nov 1, 2023.
[17] Stefano Dalla Palma, Dario Di Nucci, Fabio Palomba, and Damian Andrew Tamburri. 2020. Toward a catalog of software quality metrics for infrastructure code. *Journal of Systems and Software* 170 (2020), 110726.
[18] Stefano Dalla Palma, Dario Di Nucci, and Damian A Tamburri. 2020. AnsibleMetrics: A Python library for measuring Infrastructure-as-Code blueprints in Ansible. *SoftwareX* 12 (2020), 100633.
[19] GitHub. [n. d.]. The State of Open Source and AI. https://github.blog/2023-11-08-the-state-of-open-source-and-ai/. Accessed on: Nov 12, 2023.
[20] hashicorp. [n. d.]. https://www.terraform.io/. Accessed on: Nov 1, 2023.
[21] Red Hat. [n. d.]. https://www.ansible.com/. Accessed on: Nov 1, 2023.
[22] Richard Helm, Ralph E Johnson, Erich Gamma, and John Vlissides. 2000. *Design patterns: Elements of reusable object-oriented software*. Braille Jymico Incorporated Quebec.
[23] Pandu Ranga Reddy Konala, Vimal Kumar, and David Bainbridge. 2023. SoK: Static Configuration Analysis in Infrastructure as Code Scripts. In *2023 IEEE International Conference on Cyber Security and Resilience (CSR)*. IEEE, 281–288.
[24] Luigi Lavazza, Sandro Morasca, and Marco Gatto. 2023. An empirical study on software understandability and its dependence on code characteristics. *Empirical Software Engineering* 28, 6 (2023), 1–24.
[25] Valentina Lenarduzzi, Terhi Kilamo, and Andrea Janes. 2023. Does Cyclomatic or Cognitive Complexity Better Represents Code Understandability? An Empirical Investigation on the Developers Perception. *arXiv preprint arXiv:2303.07722* (2023).
[26] Alisdair McDiarmid. [n. d.]. HCL Native Syntax Specification. https://github.com/hashicorp/hcl/blob/main/hclsyntax/spec.md. Accessed on: Nov 1, 2023.
[27] Kief Morris and Brice Thompson. 2020. *Infrastructure as Code* (2nd ed.). O'Reilly Media.
[28] Ruben Opdebeeck, Ahmed Zerouali, and Coen De Roover. 2023. Control and data flow in security smell detection for infrastructure as code: Is it worth the effort?. In *2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR)*. IEEE, 534–545.
[29] Sneh Pandya and Riya Guha Thakurta. 2022. Hands-on Infrastructure as Code with Hashicorp Terraform. In *Introduction to Infrastructure as Code: A Brief Guide to the Future of DevOps*. Springer, 99–133.
[30] Perforce. [n. d.]. https://www.puppet.com/. Accessed on: Nov 1, 2023.
[31] Anthony Peruma, Khalid Almalki, Christian D Newman, Mohamed Wiem Mkaouer, Ali Ouni, and Fabio Palomba. 2020. Tsdetect: An open source test smells detection tool. In *Proceedings of the 28th ACM joint meeting on european software engineering conference and symposium on the foundations of software engineering*. 1650–1654.
[32] Akond Rahman, Md Rayhanur Rahman, Chris Parnin, and Laurie Williams. 2021. Security smells in ansible and chef scripts: A replication study. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 30, 1 (2021), 1–31.
[33] runterrascan. [n. d.]. https://github.com/tenable/terrascan. Accessed on: Nov 1, 2023.
[34] Nuno Saavedra and João F Ferreira. 2022. GLITCH: an Intermediate-Representation-Based Security Analysis for Infrastructure as Code Scripts. *arXiv preprint arXiv:2205.14371* (2022).
[35] scitools. [n. d.]. Understand Software Metrics. https://documentation.scitools.com/pdf/metricsdoc.pdf. Accessed on: Nov 1, 2023.
[36] Tushar Sharma. [n. d.]. Smell detection tool for Puppet code. https://github.com/tushartushar/Puppeteer. Accessed on: Nov 1, 2023.
[37] Tushar Sharma, Marios Fragkoulis, and Diomidis Spinellis. 2016. Does your configuration code smell?. In *Proceedings of the 13th International Conference on Mining Software Repositories*. 189–200.
[38] Dave Smith. [n. d.]. https://gitlab.com/gitlab-com/gl-infra/production/-/issues/15999. Accessed on: Nov 1, 2023.
[39] SonarSource. [n. d.]. https://github.com/SonarSource/sonar-iac/tree/master/iac-extensions/terraform. Accessed on: Nov 1, 2023.
[40] Anthony J Viera, Joanne M Garrett, et al. 2005. Understanding interobserver agreement: the kappa statistic. *Fam med* 37, 5 (2005), 360–363.