On the Understandability of Design-Level Security Practices in Infrastructure-as-Code Scripts and Deployment Architectures

EVANGELOS NTENTOS, NICOLE ELISABETH LUEGER, GEORG SIMHANDL, and UWE ZDUN, University of Vienna, Vienna, Austria
SIMON SCHNEIDER, RICCARDO SCANDARIATO, and NICOLÁS E. DÍAZ FERREYRA, Institute of Software Security, Hamburg University of Technology, Hamburg, Germany

Infrastructure as Code (IaC) automates IT infrastructure deployment, which is particularly beneficial for continuous releases, for instance, in the context of microservices and cloud systems. Despite its flexibility in application architecture, neglecting security can lead to vulnerabilities. The lack of comprehensive architectural security guidelines for IaC poses challenges in adhering to best practices. We studied how developers interpret IaC scripts (source code) in two IaC technologies, Ansible and Terraform, compared to semi-formal IaC deployment architecture models and metrics regarding design-level security understanding. In a controlled experiment involving ninety-four participants, we assessed the understandability of IaC-based deployment architectures through source code inspection compared to semi-formal representations in models and metrics.

We hypothesized that providing semi-formal IaC deployment architecture models and metrics as supplementary material would significantly improve the comprehension of IaC security-related practices, as measured by task *correctness*. Our findings suggest that semi-formal IaC deployment architecture models and metrics as supplementary material enhance the understandability of IaC security-related practices without significantly increasing *duration*. We also observed a significant correlation between task *correctness* and *duration* when models and metrics were provided.

CCS Concepts: • **Software and its engineering → Software architectures**; **Distributed systems organizing principles**; • **Security and privacy → Software security engineering**;

Additional Key Words and Phrases: Infrastructure as code, modeling, best practices, controlled experiment, empirical software engineering

ACM Reference format:
Evangelos Ntentos, Nicole Elisabeth Lueger, Georg Simhandl, Uwe Zdun, Simon Schneider, Riccardo Scandariato, and Nicolás E. Díaz Ferreyra. 2024. On the Understandability of Design-Level Security Practices in Infrastructure-as-Code Scripts and Deployment Architectures. *ACM Trans. Softw. Eng. Methodol.* 34, 1, Article 6 (December 2024), 37 pages.
https://doi.org/10.1145/3691630

Authors' Contact Information: Evangelos Ntentos (corresponding author), Research Group Software Architecture, Faculty of Computer Science, University of Vienna, Vienna, Austria; e-mail: evangelos.ntentos@univie.ac.at; Nicole Elisabeth Lueger, University of Vienna, Faculty of Computer Science, Software Architecture Group, Doctoral School Computer Science, Vienna, Austria; e-mail: nicole.lueger@univie.ac.at; Georg Simhandl, Research Group Software Architecture, Faculty of Computer Science, University of Vienna, Vienna, Austria; e-mail: georg.simhandl@univie.ac.at; Uwe Zdun, Research Group Software Architecture, Faculty of Computer Science, University of Vienna, Vienna, Austria; e-mail: uwe.zdun@univie.ac.at; Simon Schneider, Institute of Software Security, Hamburg University of Technology, Hamburg, Germany; e-mail: simon.schneider@tuhh.de; Riccardo Scandariato, Institute of Software Security, Hamburg University of Technology, Hamburg, Germany; e-mail: riccardo.scandariato@tuhh.de; Nicolás E. Díaz Ferreyra, Institute of Software Security, Hamburg University of Technology, Hamburg, Germany; e-mail: nicolas.diaz-ferreyra@tuhh.de.

ACM 1557-7392/2024/12-ART6
https://doi.org/10.1145/3691630

1 Introduction

In the rapidly progressing field of software development, **Infrastructure as Code (IaC)** has emerged as an essential paradigm, significantly altering how IT infrastructure is provisioned and managed. Unlike traditional methods, IaC introduces the concept of managing infrastructure through code, enabling dynamic, and automated environments [1]. This approach enhances operational efficiency and aligns with contemporary development practices focusing on agility and scalability [2, 3].

The adoption of IaC has been driven by the increasing complexity of deploying microservice-based and cloud-based systems, along with the pressing demand for rapid releases. The difference between conventional deployment architectures and IaC-based ones lies in automation and repeatability. Conventional infrastructure management struggles to keep pace with the swiftly evolving demands of these systems, necessitating a shift towards more agile and flexible practices. In contrast, IaC automates infrastructure provisioning and configuration through code, ensuring consistency, scalability, and faster deployment cycles across diverse environments with minimal manual intervention. This paradigm shift allows development teams to employ the same principles of versioning, testing, and deploying to infrastructure as they do with software applications, significantly reducing the time and effort required for infrastructure management.

IaC architecture defines infrastructure using code that specifies the desired state of the infrastructure. The code is versioned using a source control system like Global Information Tracker, which allows teams to collaborate on infrastructure changes and track changes over time. **Continuous Integration/Continuous Delivery (CI/CD)** pipelines automate the testing, building, and deploying of infrastructure code changes. This automation streamlines the infrastructure-related processes and ensures that infrastructure is consistent, repeatable, and scalable. Moreover, IaC can help improve security by providing visibility into infrastructure changes and enabling teams to identify and fix security vulnerabilities [1, 4].

According to [5], Terraform[1] and Ansible,[2] the two technologies we study, collectively account for 50% of the usage among IaC tools. These technologies are widely adopted for infrastructure management and deployment automation. Terraform is an open-source IaC tool that helps manage infrastructure declaratively and can be used across multiple cloud providers. Ansible is also an open-source tool that automates tasks across multiple servers.

1.1 Problem Statement

Despite its benefits, the flexibility of IaC can lead to security issues. This could expose vulnerabilities that attackers could exploit to access procedures and run code to compromise the application [6–8]. While some works address IaC security issues at the implementation level [9–11], design- or architecture-level security is not yet the focus of academic studies. This is surprising, as insecure software design has garnered increased attention from the cybersecurity community in recent years [12]. Notably, the latest Top 10 most critical vulnerabilities of **Open Worldwide Application Security Project (OWASP)** have introduced a new category entitled "Insecure Design" [13], highlighting flaws in a software system's overall architecture or design. This underscores the importance of addressing architectural security issues to mitigate risks and safeguard systems against potential threats. So far, no works have studied the comprehensibility of IaC models for design-level security.

With design-level security, we refer to vulnerabilities that are inherent to the overall architecture or structure of a system rather than specific implementation details. In IaC, while scripts are indeed

[1] https://www.terraform.io
[2] https://www.ansible.com

programs, design-level issues encompass broader concerns such as how components interact, how privileges are managed, and overall system resilience. These aspects can significantly impact security but may not be adequately covered by addressing individual lines of code.

Unfortunately, the documented architectural patterns or formal guidelines that reflect security best practices in the context of IaC are limited [10, 14, 15], and relevant guidelines by industry organizations [16–21] often remain vague. Industry-scale systems support many security-related practices, each with many implementation options, making it challenging to assess whether an IaC deployment conforms to recommended best practices. Even worse, many IaC systems are used in a continuous release or CI/CD context, requiring such assessment for every commit related to IaC code.

To illustrate this point, consider the deployment architecture level concerns access and traffic control [22] in IaC scripts. To understand if an IaC script securely employs these concerns, one would not only need to study a local change to an access and traffic control definition but also related concepts such as security groups, linked components and deployment nodes, alternative paths to reach components and deployment nodes, and so on. So, for every commit, the entire IaC script must be studied.

Even though tools like Ansible and Terraform [23] provide relatively readable languages and IaC scripts can be modularized (so, most of the time, not all deployments of an organization, only the scripts of one project must be checked), constant manual checking of such design-level properties in IaC script is time-consuming and error-prone.

1.2 Research Objectives, Hypotheses, and Results

Our work has two main hypotheses which we aim to study:

H1 *We hypothesize that semi-formal deployment architecture models (based on a formal modeling method), specifically designed to provide an abstraction over IaC technologies, can help in comprehending IaC-based deployment architectures by providing a better big-picture comprehension than (continuous) manual checking in IaC scripts.* For instance, a Security Group definition and the affected component and deployment nodes must be studied to study Ingress and Egress traffic control. A model provides an abstraction over different IaC technologies and implementation options in each IaC language. The semi-formal diagram depicting the model lets developers easily inspect the paths between affected components and deployment nodes visually.

H2 *We hypothesize that formally defined metrics, precisely measuring one IaC architectural best practice's usage, can help easily assess an architectural best practice and thus further enhance the comprehensibility of IaC deployment architectures.* For instance, if two metrics on the Ingress and Egress traffic control best practices are automatically measured and signal before and after a commit that 100% traffic control is given in a deployment architecture, then neither the model nor the IaC source code must be studied to assess conformance to traffic control best practices.

To test these hypotheses, we studied three formally modeled **Architectural Design Decisions (ADDs)**, commonly applied in IaC-based systems, which have 12 design-level best practices in IaC as decision options and are derived by an in-depth study of the gray literature and 21 IaC-based open-source systems developed by practitioners [24]. We study them in two open-source systems, one based on Terraform and one on Ansible. These backgrounds are explained in Section 2.

To aid developers with models and metrics, as suggested in our hypotheses, our work introduces (in Section 3):

(1) a formal modeling method for enhancing UML-based, semi-formal deployment architecture diagrams with specifics to model those decision options commonly found in the IaC script and abstract from the different IaC technologies studied in our prior study [24];
(2) one formally defined metric per decision option [24] to automatically measure its presence in a given deployment architecture.

We hypothesize that providing system models and metrics would enhance understanding IaC security practices as measured through task *correctness* and *duration*. Our results indicate that participants who were given a system source code repository, metrics, and semi-formal architectural deployment models demonstrated a significantly better understanding than those who only received the system source code repository. Providing these additional materials significantly improved the effectiveness of understanding the IaC security practices without negatively impacting the efficiency. Overall, our work contributes to filling the gap in empirical research on architectural security issues specific to IaC and deployment architectures.

1.3 Structure of This Article

We provide an overview of security ADDs in IaC-based deployments and background on Ansible and Terraform in Section 2. Section 3 explains our formal modeling method, how it is mapped to UML diagrams, and our suggested metrics. Next, we describe the planning of this study in Section 4. In Section 5, we detail the execution of the experiment, and the results are presented in Section 6. We discuss the outcomes in Section 7. Section 8 discusses related work, and Section 9 the potential threats to validity before concluding the article in Section 10.

2 Background

In this section, we summarize the background of this work, namely the architectural security decisions for IaC patterns and practices, as well as infrastructure coding languages and tools. The selection of security-related practices was informed by a thorough gray literature review and the in-depth study of 21 open-source systems in our prior work [24]. We aimed to encompass practices relevant to IaC and critical for the security of automated infrastructure management. While some practices may not appear to be unique to IaC at first glance, their application within an IaC context, especially concerning the automation and dynamism of infrastructure provisioning, offers unique challenges and opportunities for security [19].

2.1 Security in Infrastructure-as-Code Architectures

The infrastructure of a software system is a highly critical aspect of the system; thus, infrastructure automation such as IaC needs to be adequately secured. This section explains three categories of the most common security best practices or patterns for IaC architectures. Please note that many code-level or low-level security best practices for IaC have been documented [9, 10] that can be automatically checked in a few code lines, such as the use of hard-coded secrets or admin by default [9] or are more general principles such as violations of naming conventions or favoring complexity [10] that might have negative implications on security. Instead, here we focus on architecturally significant properties of the IaC architecture in the form of ADDs and options, i.e., those that need to be studied system-wide and are thus hard to detect automatically. In the following, we introduce three exemplary architectural security aspects of IaC, which we study in this article.

2.1.1 Observability in the System. An important aspect of deployment architectures is identifying and responding to what is happening within a system, what resources need to be observed, and what is causing a possible issue. Using observability practices to collect, aggregate, and analyze log

data and metrics is critical to establishing and maintaining more secure, flexible, and comprehensive systems. Moreover, collecting and analyzing information improves the detection of suspicious system behaviors or unauthorized system changes on the network and can facilitate the definition of different behavior types in which an alert should be triggered. There are several practices related to observability in the system:

— *Server Monitoring* is an essential process of observing the activity on servers, either physical or virtual [20]. Please note that this also includes all kinds of cloud nodes, e.g., nodes in a cloud cluster. A single server can support hundreds or even thousands of requests simultaneously. Ensuring that all servers operate according to expectations is critical to managing infrastructure.
— *Application Monitoring* collects log data to support aspects such as tracking availability, bugs, resource use, and application performance changes [20].
— *Metrics Collection, Services Availability, Centralized Log Management* and *Monitoring and Performance Analytics Support*: These practices can improve a system's security by observing more system features.

2.1.2 Security Access Control. A critical security factor in distributed and cloud-based systems is how stable, verifiable, and secure the interactions between a user and a cloud application are. For this, secure authentication practices address many possible issues. Authentication is the process of determining a user's identity. There are several authentication practices to secure access control:

— *Protocol-Based Authentication* uses a cryptographic protocol, **secure sockets layer (SSL)** and transport layer security to encrypt the data exchanged between a Web server and a user and provides means for authentication of the connection [17, 18].
— *Token-Based Authentication* uses a protocol that allows users to verify their identity and, in return, receive unique access tokens for a specified period [25].
— *API Keys* based authentication utilizes a unique key to authenticate a user or a calling program to an API [26]. However, they are typically used to authenticate a project with the API rather than a user.
— *Plaintext Authentication* means that the user name and password are submitted to the server in plaintext, easily visible in any intermediate router on the Internet [17].
— **Single Sign-On (SSO)** is an authentication practice that can be implemented additionally [27]. This method allows users to log into one application and access multiple applications.

Token-based Authentication and *Protocol-based Authentication* are recommended practices. The following tactics are *not* considered secure but are better than no authentication (as they provide authentication): *API Keys* are inferior as they are only static credentials and thus can be easily transferred and used in unintended scenarios. *Plaintext Authentication* is the weakest form of authentication in terms of security. *Plaintext Authentication over an Encrypted Protocol* would improve *Plaintext Authentication* as the protocol then prevents sniffing the traffic in the network to get the credentials. However, the client and server applications are still managed in plaintext. Thus, *Plaintext-based Authentication over an Encrypted Protocol* is also considered to be not secure. Where applicable, SSO is a useful technique that should be used in addition to the recommended authentication practices.

2.1.3 Traffic Control in the System. Controlling incoming and outgoing traffic in a system can significantly improve overall security. Traffic can be controlled by Gateway components through which all incoming or outgoing traffic must go. There are two common practices in the field:

—*Ingress Traffic Control* refers to traffic that enters the boundary of a network [28]. The ability to control what is entering a system is important for establishing security since it can prevent possible attacks from outside the network, where many possible attacks originate.
—*Egress Traffic Control* refers to traffic that exits a network boundary [29]. It is an important aspect of network and system security, as it helps to ensure that outgoing traffic is managed effectively and securely.

Both practices can be specified by *security rules* implementing *security groups* that act as a virtual firewall for a system.

2.2 Infrastructure Coding Languages and Tools

In this work, we study two exemplary, widely used IaC languages and tools: Ansible, which is an imperative approach, and Terraform, which is a declarative. In this section, we briefly summarize them as background.

2.2.1 Ansible. Ansible is an open-source tool that automates tasks across multiple servers. With Ansible, you can define tasks in a simple YAML format, which can be used to configure servers, deploy applications, and manage infrastructure. Ansible uses a client-less architecture, meaning it does not require any agents to be installed on the remote machines and uses SSH to connect to servers. It also has an extensive library of pre-built modules to automate various tasks. Ansible automation is used to install software, automate daily tasks, provision infrastructure, improve security and compliance, patch systems, and share automation across your teams.

2.2.2 Terraform. Terraform is an open-source IaC tool that supports defining and managing infrastructure declaratively using a high-level configuration language. With Terraform, you can define the desired state of your infrastructure and then apply those changes to a cloud provider such as **Amazon Web Services (AWS)**, Google Cloud, or Azure. Terraform uses a state file to track changes and ensure the infrastructure is in the desired state. It provides a unified way to manage resources across multiple cloud providers and helps to automate the provisioning and configuration of infrastructure. It's a cloud-agnostic, open-source provisioning tool.

3 Modeling Method and Metrics

Our hypotheses revolve around aids provided as semi-formal UML diagrams (based on a formal modeling method) and formally defined metrics. We define one metric to measure the presence or absence of each decision option introduced in the previous section. In this section, we provide an overview of these aids provided to the experimental groups in our study.

3.1 Models of IaC Systems

Our proposed modeling approach utilizes a deployment model based on elements derived solely from the typical scripts employed by IaC technologies. This deployment model inherently encapsulates the application's architecture slated for deployment, facilitating the assessment of adherence to architectural decisions. Consequently, it becomes imperative to operate with minimal elements, as parsing them from the IaC scripts could otherwise prove challenging.

An IaC-based deployment can be represented using formal model diagrams. We use and extend a formal modeling method based on our prior work [30]. We extend it here to model the integration of component and deployment nodes. A deployment architecture model M is a tuple (N_M, C_M, NT_M, CT_M, c_source, c_target, $nm\_connectors$, n_type, c_type) where:

—N_M is a finite set of component and infrastructure *nodes* in Model M.
—$C_M \subseteq N_M \times N_M$ is an ordered finite set of *connector edges*.

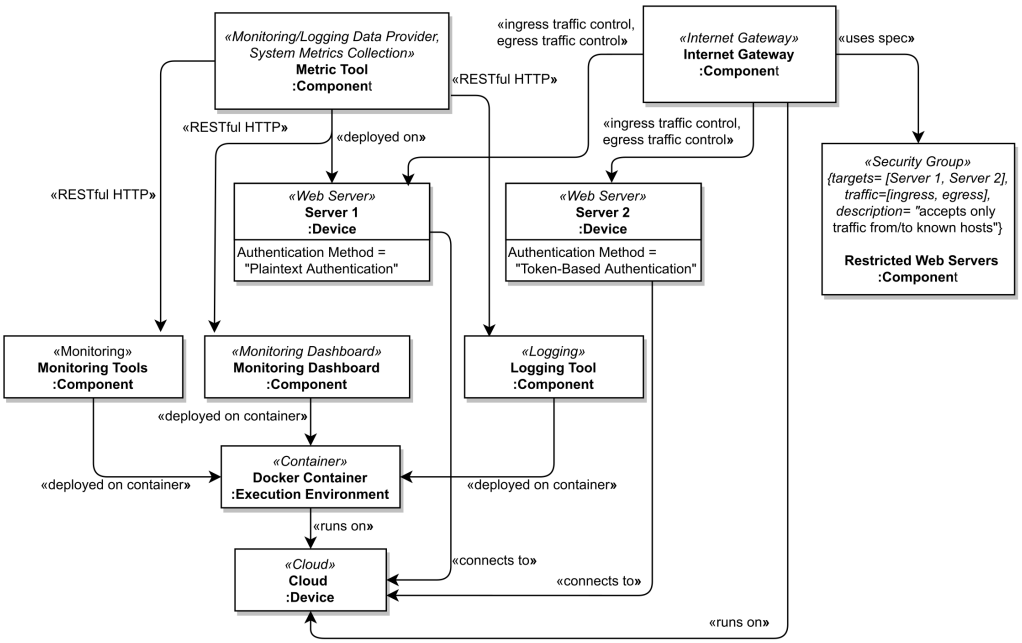

Fig. 1. Example IaC-based deployment diagram.

- NT_M is a set of *component types*.
- CT_M is a set of *connector types*.
- $c\_source : C_M \rightarrow N_M$ is a function returning the component that is the *source* of a link between two nodes.
- $c\_target : C_M \rightarrow N_M$ is a function returning the component that is the *target* of a link between two nodes.
- $nm\_connectors : \mathbb{P}(N\_M) \rightarrow \mathbb{P}(C\_M)$ is a function returning the set of connectors for a set of nodes: $nm\_connectors(nm) = \{c \in C_M : (\exists n \in nm : (c\_source(c) = n \wedge c\_target(c) \in C_M) \vee (c\_target(c) = n \wedge c\_source(c) \in C_M))\}$.
- $n\_type : N_M \rightarrow \mathbb{P}(NT_M)$ is a function that maps each node to its set of *direct and transitive node types* (for a formal definition of node types, see [30]).
- $c\_type : C_M \rightarrow \mathbb{P}(CT_M)$ is a function that maps each connector to its set of *direct and transitive connector types* (for a formal definition of connector types, see [30]).

An example of a deployment architecture model derived from this formal deployment model is shown in Figure 1 as a UML diagram. The diagram depicts various deployment nodes, deployed components, and their relations. All deployment nodes are of type *Deployment Node*, which has the subtypes *Execution Environment* and *Device*. These have further subtypes, such as *Virtual Machine (VM)* and *Container* for *Execution Environment*, and *Server*, *IoT Device*, *Cloud*, etc. for *Device*. Note that we use the function *n_type* and the component type hierarchy formally defined based on these types (see [30] for the formal definition) to model component and deployment nodes. These translate to component stereotypes in the semi-formal UML diagram.

The connector type *deployed on* is used to denote a deployment relation of a *Component* (as a connector source) on an *Execution Environment* (as a connector target). It is also used to denote the transitive deployment relation of *Execution Environments* on other ones, such as a *Container* is

deployed on a *VM*. The connector type *runs on* models the relations between execution environments and the devices they run on. Note that we use the function *c_type* and the connector type hierarchy formally defined based on these types (see [30] for the formal definition) to model connector relations specific to such deployment architectures. These translate to connector stereotypes in the semi-formal UML diagram.

This example diagram shows the relations of *Components*, *Deployment Nodes*, and *Execution Environments* of an IaC-based deployment architecture. Here, a *Metric Tool* is deployed on a *Web Server*, which is a *Device*, and connected to a *Monitoring Dashboard* and a *Monitoring Tools* component. The *Web Server* has attributes specifying the *authentication method* it uses (e.g., Plaintext, Token-based). The monitoring components are deployed on an *Execution Environment*, here a Docker Container. An *Execution Environment* runs on a *Device*, e.g., here a *Cloud* node. Moreover, an *Internet Gateway* also runs on the *Cloud*, which controls the traffic going into and out of the two deployed servers.

This deployment architecture contains several Observability (security) practices, namely Monitoring, Logging, and System Metrics Collection for Server 1, but no Observability components are deployed for the second server. For both servers, Traffic Control is provided through an Internet Gateway (of the cloud provider) that uses a Security Group to specify restricted Ingress and Egress traffic for the servers.

3.2 Metrics

In our previous work [24], we have developed a comprehensive set of generic, technology-agnostic metrics for security-related decisions made in IaC-based deployment architectures, ensuring that each of the twelve options of our three ADDs (see Section 2.1) is associated with at least one metric. These metrics each evaluate conformance to one particular design-level IaC security practice. In [24], by establishing a ground truth, we objectively assessed the level of support for patterns and practices within 21 open-source case study systems and statistically validated those metrics. Defining technology-independent metrics enables automated numerical assessment of pattern or best practice implementations across different deployment models and IaC implementation technologies. Our prior work accurately predicted the ground truth assessment using ordinal regression analysis with these metrics as independent variables. Our findings indicate that decision-related metrics can reliably predict our objective evaluations, suggesting the feasibility of an automated, metrics-based assessment of a system's compliance with ADD options with a high level of certainty.

The metrics have continuous values ranging from 0 to 1, with 1 representing the optimal case where a set of patterns or best practice is fully supported and 0 the worst-case scenario where it is absent. The Plaintext Authentication utilization metric is an exception as in this metric; the scale is reversed compared to the others because here we detect the presence of an anti-pattern: the optimal result of our metrics is 0, and 1 is the worst-case result. All metrics used in this study are summarized in Tables 1–3. Please note that there is one metric for indication of support of each ADD decision option presented in Section 2.1.

4 Experiment Planning

Our study adheres to the empirical research guidelines for software engineering proposed by Jedlitschka et al. [31]. In addition, the study design incorporates guidelines for empirical research in software engineering from Kitchenham et al. [32], Wohlin et al. [33], and Juristo and Moreno [34]. For statistical analysis of the collected data, the study uses the robust statistical method guidelines for empirical software engineering by Kitchenham et al. [35].

Table 1. Observability in the System

Metric Name	Description
Server Monitoring Ratio (SEM)	This metric returns the server nodes that support server monitoring divided by the total number of server nodes.
Application Monitoring Ratio (AMS)	This metric returns the server nodes on which application-level monitoring is deployed divided by the total number of server nodes.
System Metric Collection Ratio (SMC)	This metric returns the server nodes on which security-related metrics are collected divided by the total number of server nodes.
Centralized Log Management Ratio (CLM)	This metric returns the server nodes on which a centralized log management solution is deployed divided by the total number of server nodes.
Service Availability Monitoring Ratio (SAS)	This metric returns the server nodes on which a dedicated service availability monitoring solution is deployed divided by the total number of server nodes.
Performance Analytics Support Ration (PAS)	This metric returns the server nodes on which a service performance analytics monitoring solution is deployed divided by the total number of Server Nodes.

Table 2. Security Access Control

Metric Name	Description
SSL Protocol-Based Authentication Utilization Ratio (SSLA)	The SSLA metric returns the proportion of server nodes that support SSL Protocol-based Authentication.
Token-Based Authentication Utilization Ratio (TBA)	The TBA metric returns the proportion of server nodes that support Token-Based Authentication.
Plaintext Authentication Utilization Ratio (PLA)	The PLA metric returns the proportion of server nodes that support Plaintext Authentication.
API Keys Utilization Ratio (API)	The API metric returns the proportion of server nodes that support Authentication via API Keys.
Single Sign-On Utilization Ratio (SSO)	The SSO metric returns the proportion of server nodes that support SSO in their authentication method.

Table 3. Traffic Control in the System

Metric Name	Description
Ingress Traffic Control Utilization Metric (ING).	The ING metric returns the proportion of server nodes for which the ingress (incoming) traffic is traffic-controlled.
Egress Traffic Control Utilization Metric (EGR).	The EGR metric returns the proportion of server nodes for which the egress (outgoing) traffic is traffic-controlled.

4.1 Goals

This experiment aims to perform a code review of one small and one medium-sized IaC deployment architecture (one realized with Ansible, one with Terraform) in a limited time session of 1.5 hours regarding the recommended security design patterns and practices. The focus is on the review of specific recommended security design patterns and practices. In particular, the Observability, Authentication, and Traffic Control practices explained before will be studied. For both systems, the participants reviewed a system source code repository (including a short documentation in a README file). To study how far models and metrics can help participants understand a given system, each participant received help in the form of a model of the IaC system and related metrics to be studied in one of the two tasks. In the tasks where this help is present, we suggest consulting the models and metrics and then studying the code if more details are required. We use metrics and models based on the methods introduced in the paper [24].

4.2 System Description

The selection of the systems is based on their distinctive approaches and technologies used for deploying and managing cloud-based services, focusing on security, monitoring, and deployment and their good representation of the security practices investigated in our approach. We selected them as they represent major approaches regarding the three studied ADDs according to our study of numerous gray literature sources and an in-depth study of 21 open-source systems in our prior work [24].

The Ansible-based system emphasizes cloud security and efficient network management through an **Elasticsearch, Logstash, Kibana (ELK)** server for VM monitoring and configuration change detection. This system showcases a securely monitored and well-organized network architecture, focusing on security and monitoring. Please note that this is a commonly used tutorial example. We have chosen an implementation that realizes the necessary practices and not much more to avoid participants having to search through the source code for the relevant parts[3] The Terraform system is larger and more complex than the Ansible example. It illustrates the deployment of microservices to AWS, showcasing a simple yet effective architecture for deploying a frontend and a backend server with service discovery, focusing on deployment simplicity and microservice architecture. More specifically:

The **Ansible-based Cloud Security (ACS)** System with ELK Stack[4] (Figure 2), incorporates an ELK server for effective VM monitoring and configuration change detection. Additionally, it employs Filebeat to capture file system data and Metricbeat to collect machine metrics, enhancing overall system monitoring. The system includes detailed instructions for utilizing the Ansible build and streamlining deployment and configuration processes to ensure efficient network management. In essence, the system represents a securely monitored and well-organized network architecture.

The **Terraform Microservice Deployment (TMD)** System[5] (Figure 3) serves as a straightforward illustration of deploying two microservices to AWS using Terraform. These microservices, a frontend web server (utilizing Next.js) and a backend API server (built with Flask), establish communication through service discovery.

Please note that we selected relatively small yet realistic systems to avoid potential bias through fatigue effects in our controlled experiment. Further, smaller systems can minimize the learning curve for participants, particularly if they are unfamiliar with the specific IaC tools used in the experiment. Finally, participants will likely be more motivated when the tasks are achievable within a reasonable timeframe. Please also note that IaC parts are usually modularized in the larger-scale system we have studied. This means the fraction of the IaC code that must be studied to assess a particular security ADD is not larger than our example IaC scripts. However, in such larger systems, finding the correct IaC scripts in the code is usually harder. We did not want to study whether participants could locate the correct IaC script, but rather the understandability of a particular script was another reason for selecting relative systems.

4.3 Context and Design

We randomly divided the participants into A1, A2, B1, and B2 groups. We used a Within-Subjects Design [36] in which the same person tests all the conditions. Each student got two tasks, one with help and one without. The two sub-tasks are called:

[3]Other considered examples are: https://github.com/sadsfae/ansible-elk, https://garutilorenzo.github.io/ansible-collection-elk/, https://github.com/dmccuk/ansible_ELK, https://github.com/DanielBerman/ansible-elk-playbook
[4]Ansible system source code available at: https://github.com/babtunee/azure-cloud-security-architecture/tree/master
[5]Terraform system source code available at: https://github.com/ryanmcdermott/terraform-microservices-example

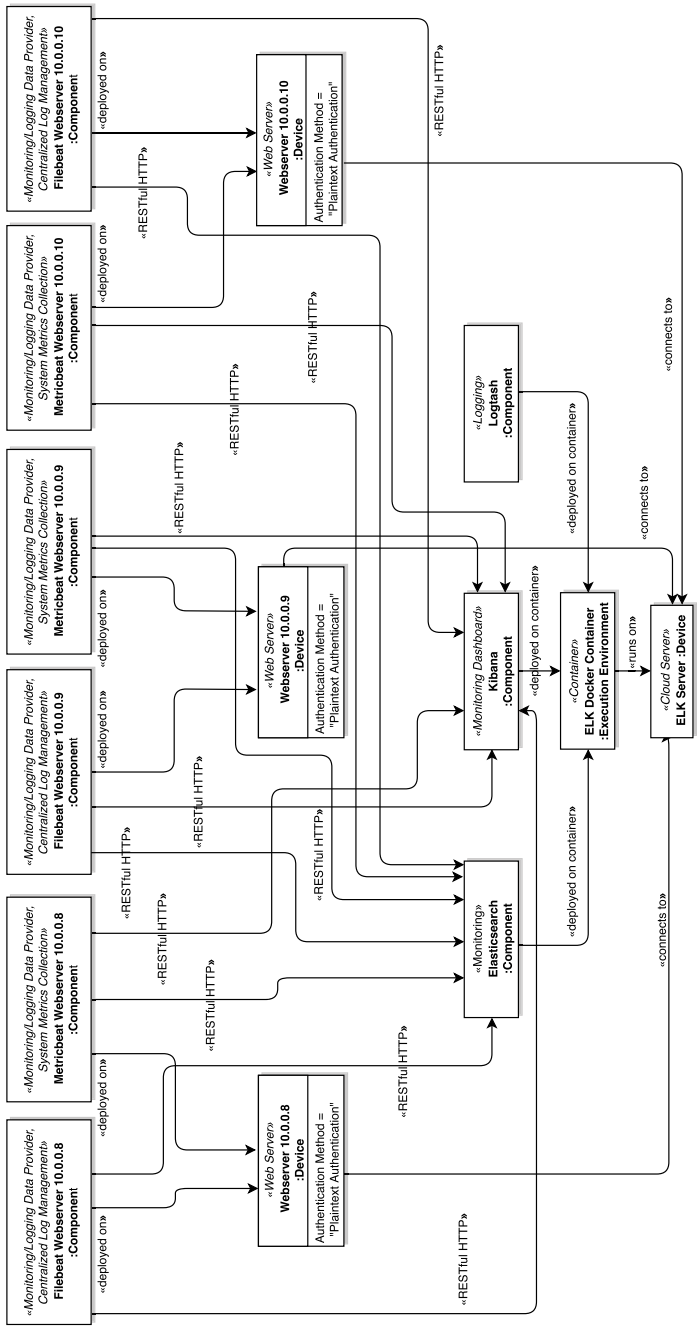

Fig. 2. Semi-formal model of the ACS system with ELK stack.

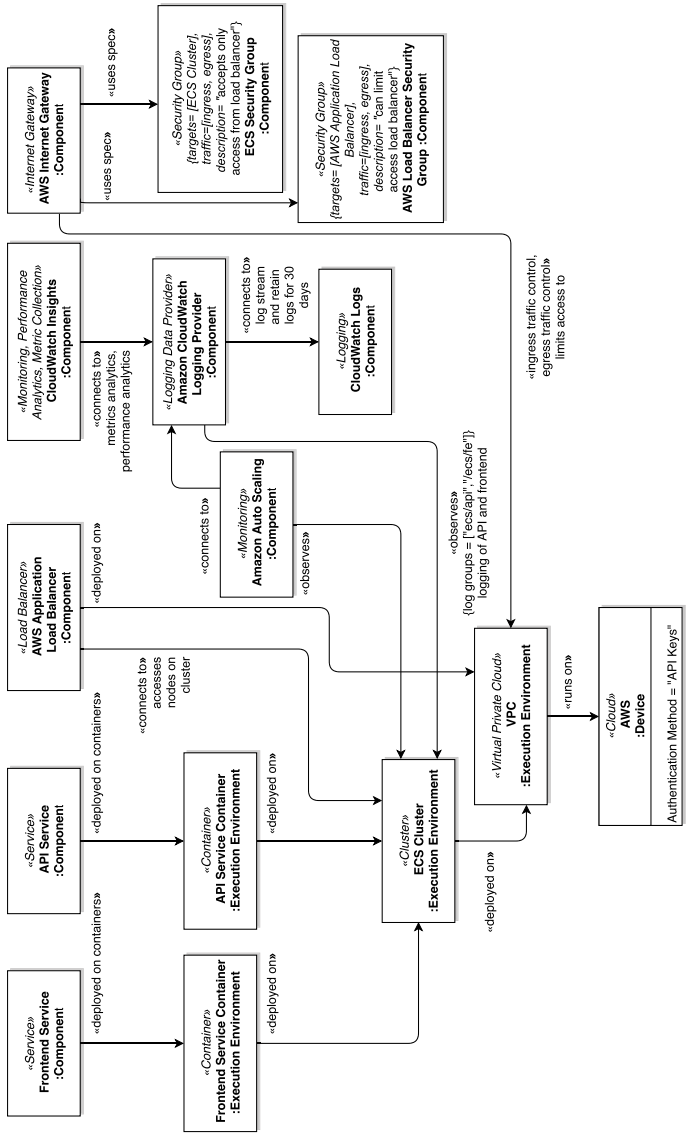

Fig. 3. Semi-formal model of the TMD system.

—ACS Using the ELK Stack.
—TMD.

The groups are composed as follows:

—(A1) ACS: system source code repository and additional help with formal models and metrics + TMD: only system source code repository.
—(A2) TMD: only system source code repository + ACS: system source code repository and additional help with formal models and metrics—identical to A1, only order reversed.
—(B1) TMD: system source code repository and additional help with formal models and metrics + ACS: only system source code repository.

—(B2) ACS: only system source code repository + TMD: system source code repository and additional help with formal models and metrics—identical to B1, only order reversed.

We added the reserved-order groups A2 and B2 to avoid possible biases from the tasks with or without help. For example, participants might spend more time on the first task, or there could be some learning effect in the general code-reviewing activity. Henceforth, we call the tasks in the four groups, where the participants get only a system source code repository, the **Control Tasks (CT)**, and the tasks in the four groups, where the participants get a system source code repository plus help in the form of formal metrics and models, the **Experiment Tasks (ET)**.

4.4 Participants

Our participants are advanced Bachelor's and some Master's students from the course **Software Engineering 2 (SE2)**. The students received 5% of the course points for participation in this experiment (a hands-on task in the lecture). Other than that, there were no additional incentives for participation. Students could participate in the lecture task but opt out of participation in the experiment. In the lectures before this hands-on task, we discussed code review, design practices, and general design patterns and practices. Participants received an introduction text[6] in which they got introduced to backgrounds like IaC, relevant IaC security practices, the modeling techniques used in the experiment, and the metrics used. Additionally, the SE2 course features an introductory pre-test to evaluate students' readiness for the course. Functioning as a diagnostic instrument, the pre-test enables us to pinpoint any knowledge gaps and customize the course content to address specific areas of concern.

Furthermore, our participants exhibit diverse knowledge and expertise in software development. For instance, a comparison of their characteristics with those from a 2016 survey involving fifty thousand developers on the online platform Stack Overflow[7] reveals striking similarities in key demographics when compared to our own participants. Significantly, all participants in our study had diverse programming experience, a crucial factor for efficiently understanding and reviewing the source code materials (refer to Section 6.2 for details).

Our study aims do not focus on techniques that only a few highly trained experts can apply, so we did not specifically target such individuals. Instead, we focus on evaluating the use of techniques by software engineers who are not experts in all aspects of the study (like software developers who apply IaC techniques occasionally and need to review the security of their software, even though they are not security experts) or even novice software engineers. Following Kitchenham et al. [32] findings, using students in our study is appropriate because they are the next generation of software professionals and are thus similar to the population of interest. This is reinforced by the fact that some students who participated in our experiment had some years of programming and industry experience. Other studies, such as those by Höst et al. [37], Runeson [38], Svahnberg et al. [39], Salman et al. [40], and Falessi et al. [41], argue that students can be valid representatives for professionals in specific empirical software engineering experiments.

4.5 Material and Tasks

The experiment is based on a selection of security-related practices. The selection includes the practices for *Observability in the System*, *Security Access Control*, and *Traffic Control in the system* summarized in Section 2.1.

[6]See the *Information Sheet* included in the experiment documents provided online in a long-term archive at: https://doi.org/10.5281/zenodo.10958738

[7]https://insights.stackoverflow.com/survey/2016

The tasks designed for the experiment were carefully chosen to reflect real-world scenarios where understanding components' deployment and operational context is crucial. This approach aligns to evaluate the effectiveness of IaC-based architectural model diagrams and metrics in enhancing the understandability of security practices. It is important to highlight that participants received an introduction and were allowed to consult a preparatory document providing necessary background information. This ensures they were not disadvantaged due to a lack of familiarity with the technology or the specific tools mentioned. This preparatory phase aimed to level the playing field and allow an assessment focused on the added value of diagrams and metrics for understandability.

The selected software practices are related to the subjects taught in the course SE2, such as related software modeling techniques, software design patterns, and software architecture. This study consists of two major experiment material artifacts[8]:

(1) *Information Sheet*. A document explaining the IaC security practices with an example model and the related metrics was provided to participants two weeks before the experiment was conducted.
(2) *Survey Form*. Four experiment survey forms per group were handed out to participants during the experiment.

All experiment survey forms are structured the same way, consisting of three parts: (1) a participant information questionnaire; (2) two ET (for one of the four groups A1, A2, B1, B2 explained in Section 4.3); (3) an overall experiment questionnaire. Every task is divided into sub-tasks to test the participants' understandability of security-related practices. The participants were instructed to read the code and descriptions in the given repositories for each system before they started to process the following four sub-tasks:

—Four tasks on listing model elements (components and connectors) were used to determine the understanding of system structure and relations that are relevant for understanding the security practices at the architectural level. An example question in Task TMD.2 is: "*List the server nodes of service components and/or AWS system components for which logging via Amazon CloudWatch is configured.*" In such questions, participants should demonstrate an understanding of more complex, non-local aspects of security architecture practices. For instance, in this example, participants must understand that the CloudWatch observability is configured via the CloudWatch Logging Provider, which observes two log groups on the ECS cluster. These log groups correspond to the two deployed services, Frontend Service and API Service. It might further confuse participants that they are not directly deployed on the cluster but are using a container. All these connections must be inspected during a security architecture review to ensure that all such services have a defined log group and are linked to CloudWatch. Figure 4 depicts the insights participants must uncover in various parts of the models and/or source code to address the corresponding task.
—One task with three filling-out-blanks sentences was used to determine the understanding of system structure. An example sentence in Task ACS.2: "*The ELK-Stack used for monitoring in the project is deployed in ___ number of Docker containers.*" In such questions, complex, non-local aspects of security architecture must be understood, and there is a chance of locating too many or not all components or deployment nodes that require a link. Here, we check whether participants found all required links to the ELK-Stack correctly. During a security

[8] See the experiment documents provided online in a long-term archive to enable replicability of our study at https://doi.org/10.5281/zenodo.10958738

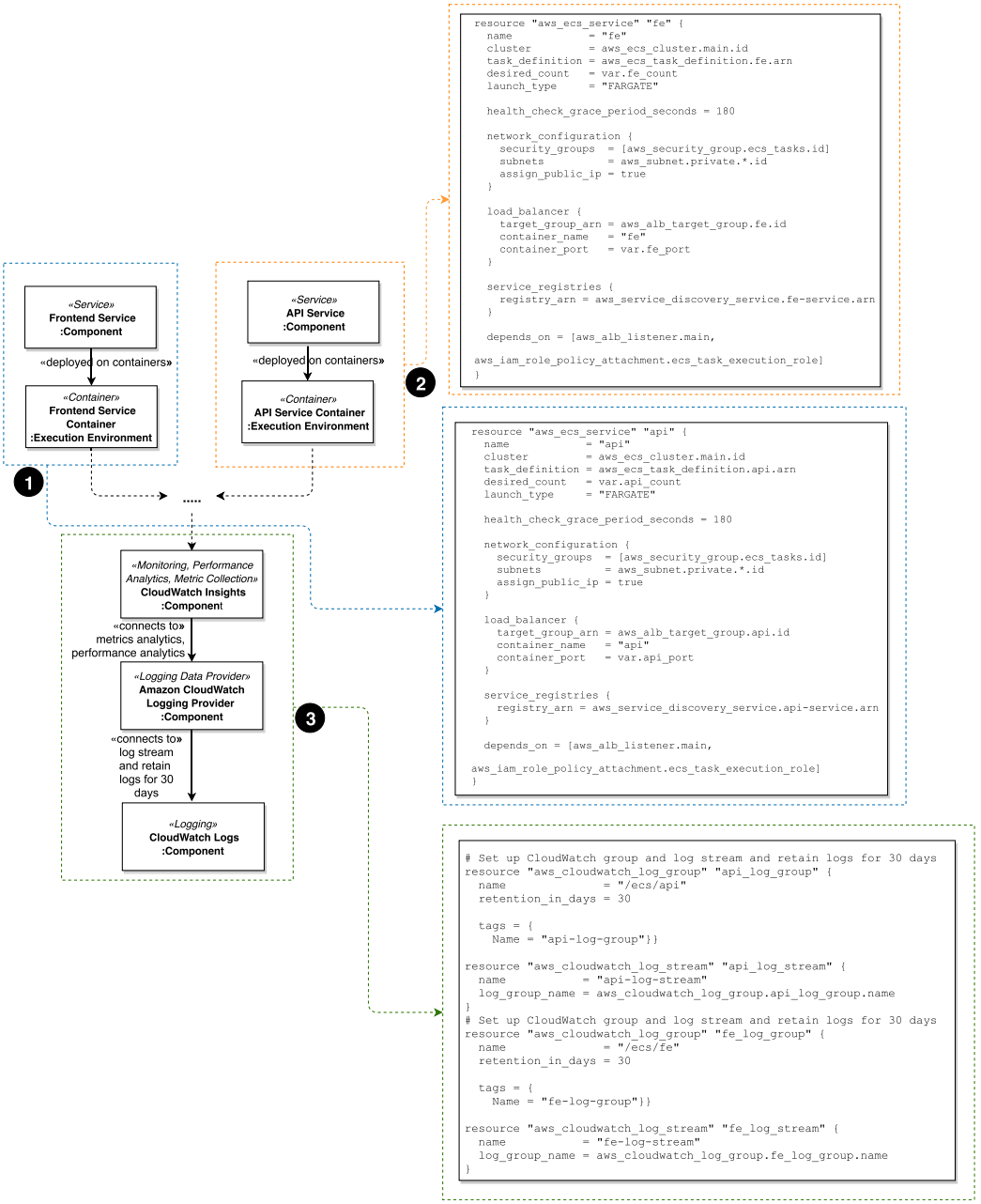

Fig. 4. Steps needed to locate the necessary source code or model components for Task TMD.1 are as follows: (1) Examine the frontend service component; (2) examine the API service component; (3) examine the Amazon CloudWatch components.

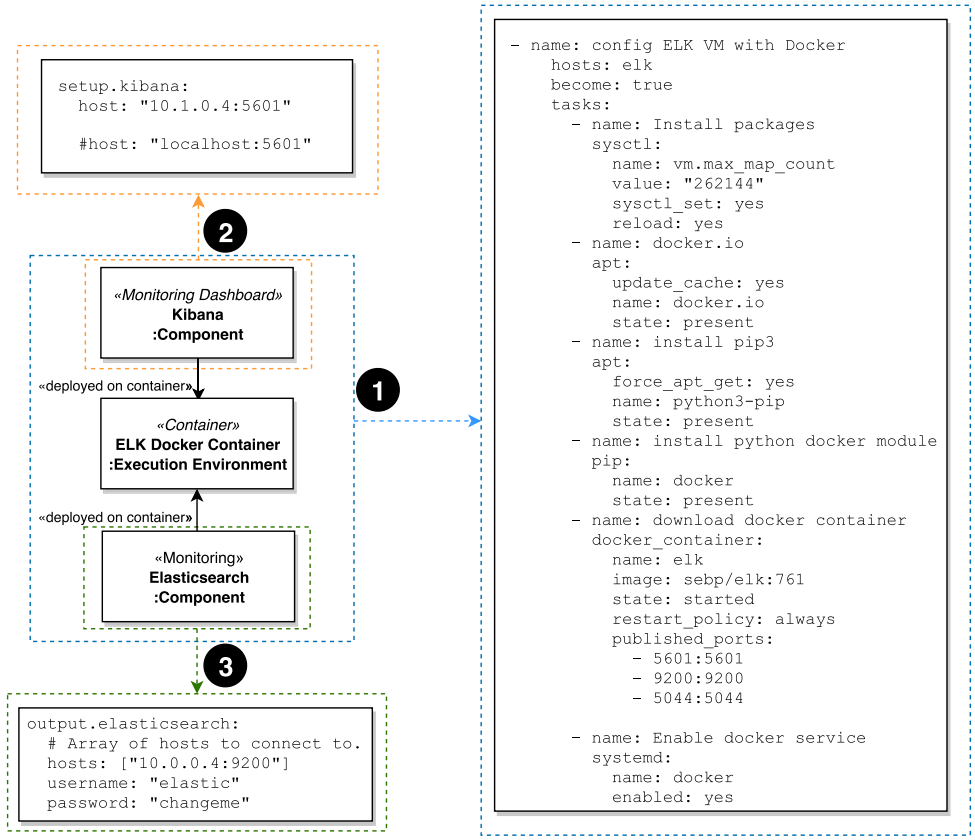

Fig. 5. Steps needed to locate the necessary source code or model components for Task ACS.2 are as follows: (1) Analyze the configuration of the ELK docker container; (2) examine the Kibana monitoring dashboard component; (3) investigate the Elasticsearch monitoring component.

architecture review, components or deployment nodes that require a link to guarantee full observability might be overlooked, leading to a possible vulnerability. Figure 5 shows the insights participants need to find in various parts of the models and/or source code to address the corresponding tasks.

— Two tasks with four True/False answers of practices were used to determine the understanding of used/supported/realized techniques in the provided systems. An example in Task TMD.1:
  – *A load balancer gets deployed by the system and is monitored using Amazon CloudWatch.*
  – *The deployed system would support centralized log management via Amazon CloudWatch, but it is disabled.*
  – *The deployed system supports monitoring the performance of services running on the monitored server nodes. For this, CloudWatch Insights can be used, a service that supports the analytics of CloudWatch logs.*
  – *Amazon auto-scaling uses the data provided by CloudWatch for Server Monitoring.*

  These questions were used to check whether participants understood the locally used security techniques and definitions, which usually required inspecting the definitions and only directly connected components or deployment nodes.

—In addition to the content tasks, a task-based questionnaire was used to obtain an objective perspective of the participants' self-assessment of how to correct their answers with a certain level of confidence.

Please note that we concentrate on questions requiring participants to analyze and sometimes synthesize the information they find in different parts of the IaC scripts or the models, where multiple implementation options often exist or understanding beyond mere memorization is required. For instance, they aim to check whether participants understand the explicit and implicit links between components and deployment nodes of the deployment architecture.

4.6 Variables and Hypotheses

This controlled experiment measures the following two dependent variables:

—*Correctness* as achieved in answering the questions, which includes trying to mark the correct answer and filling in the blanks in the tasks. It is used to measure the effectiveness of understanding the IaC security practices.
—*Duration* as the time it took to answer the questions of all tasks in the experiment survey form, excluding breaks. It is used to measure the efficiency of understanding the IaC security practices.

These two dependent variables, *correctness* and *duration*, are commonly used to measure the construct understandability [42–46].

We devised the following Null Hypotheses:

—$H_0 1$ The effectiveness of understanding the IaC security practices used shows no significant difference (similar performance) for ET compared to CT.
—$H_0 2$ The efficiency of understanding the IaC security practices used shows no significant difference (similar performance) for ET compared to CT.
—$H_0 3$ The efficiency and effectiveness of understanding the IaC security practices used show no significant difference (similar performance) for ET compared to CT.

and the corresponding Alternative Hypotheses:

—$H_a 1$ The effectiveness of understanding the IaC security practices used shows a significant difference (better performance) for ET compared to CT.
—$H_a 2$ The efficiency of understanding the IaC security practices used shows a significant difference (better performance) for ET compared to CT.
—$H_a 3$ The efficiency and effectiveness of understanding the IaC security practices used show a significant difference (better performance) for ET compared to CT.

The first hypothesis, $H_0 1$, is supported by the work documented by Vujovic et al. [47] which highlights the pivotal role of multimodal learning analytics in shaping the development of educational materials. It suggests that through the deployment of semi-formal models, these analytics can significantly enhance the effectiveness of learning by providing a more tailored and comprehensive educational experience. The second hypothesis, $H_0 2$, is supported by the work presented by Wong et al. [48], which shifts the focus to the integration of supplementary materials within online learning environments. Contrary to concerns that such materials may detract from learning efficiency, their findings reveal that they actually foster self-regulated learning. This suggests that well-designed supplementary content can complement the core material by providing learners with additional resources to manage their learning process more effectively. Finally, the third hypothesis, $H_0 3$, is supported by the work presented by Xie et al. [49], which underscores the benefits of

incorporating mobile technology and the experience-sampling method into educational settings. According to them, these technologies offer real-time insights into how students engage with the material, thereby offering a detailed perspective on the intricate relationship between the amount of time invested in learning and the resultant learning outcomes.

5 Experiment Execution

This experiment was executed in two steps: a preparation and a procedure phase.

5.1 Preparation

Two weeks before the experiment, we handed out the preparation material (the experiment information sheet, see Section 2) through an e-learning platform.[9] This document provided general information about the upcoming experiment and an introduction to the IaC technologies. This document includes all the security-related patterns and practices, the set of metrics, an example model, and a detailed description. The participants were allowed to use this document in print during the experiment. We provided the experiment information document because all participants needed to be educated to the same level of detail concerning security practices in IaC-based deployments (see Section 2).

5.2 Pilot Test

Following the preparation of our experimental materials, we conducted preliminary tests with student tutors from our research group. Similar to participants in subsequent stages, we provided the tutors with the information sheet two weeks in advance, allowing them to familiarize themselves with the necessary knowledge for the tasks. The tutors participated in the experiment under predefined conditions, which included the option to seek clarifications on the experimental procedure, a 90-minute timeframe for responding to experiment questions, and no additional support beyond the provided materials.

Beyond the pilot tests, we gathered feedback from the tutors about their experiences. Their responses indicated that we had crafted the information sheet exceptionally well, offering ample guidance for addressing the experiment questions. The tutors found the experiment easy to navigate, noting that even participants new to the subject matter and concepts would likely find it accessible, thanks to the comprehensive materials and examples provided. Given this positive feedback, we made no changes to our experiment design.

5.3 Procedure

The experiment was carried out using pen and paper as if it were a (closed book) exam and PCs with restricted access to navigate the source code repositories of the two IaC systems. Participants were allowed to bring only the preparation material to process the experiment survey form, as described in the previous Section 4. At the beginning of the experiment, every participant received a random experiment survey form (see Section 4.5). During the random handing out of the forms, we distributed about equal numbers of forms of each type (A1, A2, B1, B2 in Section 4.3). Participants were instructed to fill out and process the survey from the first page to the last page in this particular order. Furthermore, a clock with seconds granularity was projected onto a wall to provide timestamp information to the participants. They were asked to track start and stop timestamps while processing the ET. The participants' task start and stop timestamps were converted to a duration in seconds and summed up to the total duration for all tasks. To maintain the confidentiality of participant data, an

[9] https://moodle.univie.ac.at/theme/university_boost/login/index.php

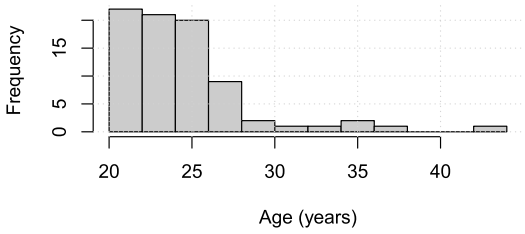

Fig. 6. Participants' age.

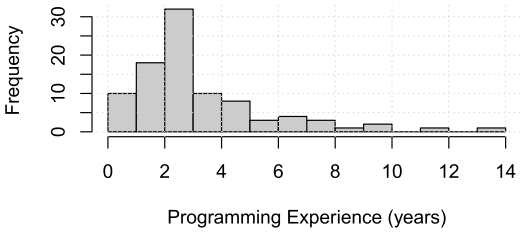

Fig. 7. Participants' programming experience.

individual who was not involved in the experiment dissociated personal information (such as name and student number) from the experiment sheets by assigning a unique identification number.

6 Analysis

The study's statistical analysis was conducted using only the R software tool.[10] The analysis involved multiple steps, such as loading the pre-processed dataset from Section 6.1, calculating descriptive statistics for the dependent variables discussed in Section 6.4, performing group-by-group comparisons utilizing appropriate statistical hypothesis tests explained in Section 6.5, and generating tables and plots for inclusion in the article. To reproduce these results, it is necessary to install the specific R library package dependencies.[11]

6.1 Dataset Preparation

The collected raw data[12] from the experiment execution phase (refer to Section 5) was prepared as follows: (1) a Microsoft Excel document was exported to a **Comma-Separated Values (CSV)** file; (2) the CSV file was imported for further processing; (3) type castings were performed for several data rows; and (4) the overall correctness for all task correctness values was calculated. The data set is published in the long-term open data archive Zenodo[13] together with all documents and R scripts.

6.2 Participant Demographics

In the study, participants exhibited a diverse demographic profile encompassing a range of ages (Figure 6), educational backgrounds (Figure 10), industry experiences (Figure 8), and programming experience (Figure 7). The age of participants varied from 20 to 40 years, indicating a mix of early-career to mid-career individuals. In terms of education (Figure 10), the experiment included individuals with **Bachelor of Science (BSc)** and **Master of Science (MSc)** degrees, as well as a subset with no higher education, reflecting a broad spectrum of academic attainment. Experience in IaC (Figure 9) was split between those with and without such experience, suggesting a variation in specific technical skills. Industry experience among participants ranged from 0 to 10 years, with a distribution that suggests both newcomers and those with a decade of experience in the workforce. Finally, programming experience also showed a wide range, spanning from 0 to 14 years, highlighting the range of technical proficiency. The participants' diverse demographic and experience profile provides a rich context for examining the impact of educational and experiential backgrounds on the study's outcomes.

[10] See https://www.r-project.org for version 4.2.2.
[11] See Data and Scripts/Scripts/install.r at https://doi.org/10.5281/zenodo.10958738
[12] See Experiment Documents/Questionnaire Results/experiment-results.csv at https://doi.org/10.5281/zenodo.10958738
[13] https://doi.org/10.5281/zenodo.10958738

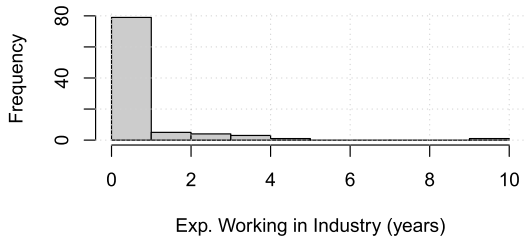

Fig. 8. Participants' software industry experience.

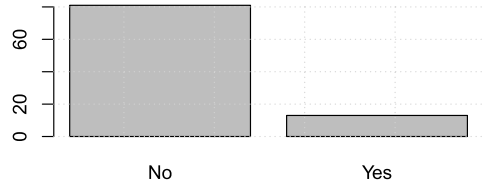

Fig. 9. Participants' IaC experience.

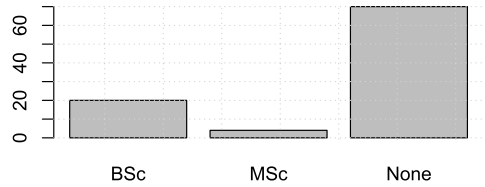

Fig. 10. Participants' education.

6.3 Normality Assessment

The normal Q-Q plots for ACS (Figure 11) for *correctness* indicate that the data for both *Control* and *Experimental* appear to follow a normal distribution. Furthermore, examining the normal Q-Q plots for TMD for *correctness* (Figure 12), it can be inferred that the data for *Control* appears normally distributed. However, it is inconclusive regarding the normality of the data for *Experimental*.

To test for normality, we chose the Shapiro-Wilk [50] normality test since, according to Razali and Yap [51], it is more powerful than alternatives (such as the Anderson-Darling [52], Lilliefors [53], and Kolmogorov-Smirnov [54]. Assuming $\alpha = 0.05$, the test for ACS indicated that the *Control*'s and *Experimental*'s distributions are not significantly different from the normal distribution, with a value $p > \alpha$. However, for TMD the test indicated that the *Control*'s distribution is also not significantly different from the normal distribution, whereas *Experimental*'s distribution significantly deviates from the normal distribution, with a value $p \leq \alpha$.

Visual inspection of the normal Q-Q plots for both groups and both systems for *duration*, visible in Figures 13 and 14, was insufficient to determine whether each group's data were normally distributed. The Shapiro–Wilk normality test for ACS indicated that *Experimental*'s distribution significantly deviates from the normal distribution. In contrast, the *Control*'s distribution is not significantly different from the normal distribution, with a value $p \leq \alpha$. However, for TMD, the test indicated that, for *duration*, the group's distribution significantly deviates from the normal distribution, with a value $p \leq \alpha$ for both groups.

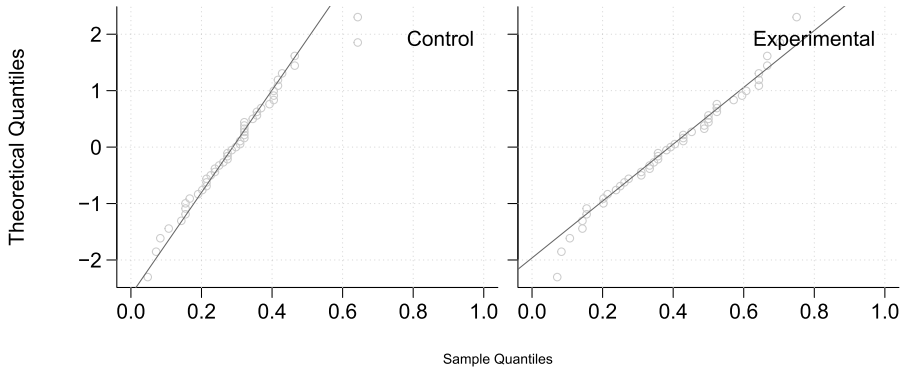

Fig. 11. Normal Q-Q plot of *correctness* (ACS).

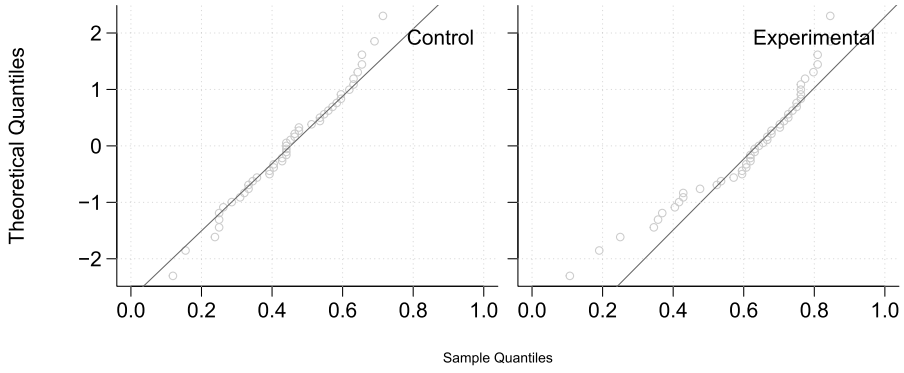

Fig. 12. Normal Q-Q plot of *correctness* (TMD).

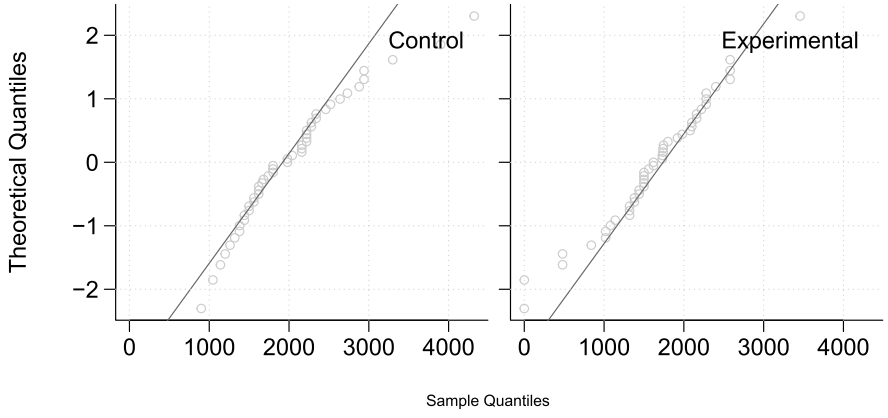

Fig. 13. Normal Q-Q plot of *duration* (ACS).

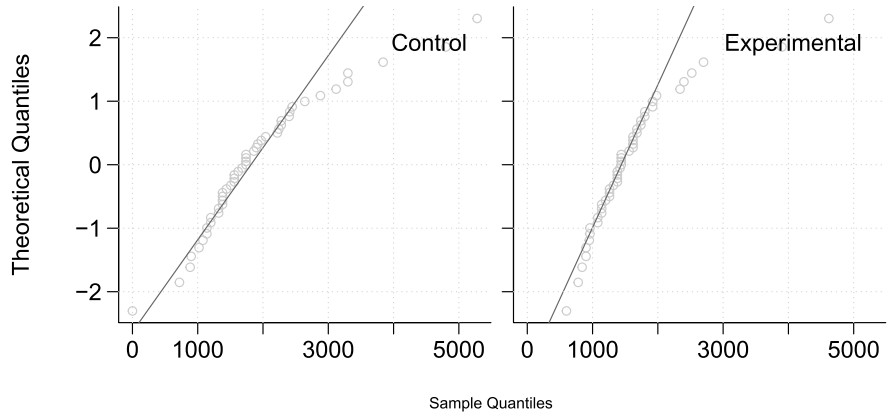

Fig. 14. Normal Q-Q plot of *duration* (TMD).

Table 4. Descriptive Statistics per Group of Dependent Variable *Correctness* (ACS)

	Control	*Experimental*
Number of observations	47	47
Mean	0.2936	0.3969
Standard deviation	0.1283	0.1816
Median	0.2976	0.3929
Median abs. deviation	0.1235	0.1942
Minimum	0.0476	0.0714
Maximum	0.6429	0.7500
Skew	0.4880	0.0167
Kurtosis	0.4827	−1.0340
Shapiro–Wilk Test p	0.1970	0.3490

Table 5. Descriptive Statistics per Group of Dependent Variable *Correctness* (TMD)

	Control	*Experimental*
Number of observations	47	47
Mean	0.4483	0.6097
Standard deviation	0.1458	0.1747
Median	0.4405	0.6429
Median abs. deviation	0.1589	0.1588
Minimum	0.1190	0.1071
Maximum	0.7143	0.8452
Skew	−0.1703	−0.9769
Kurtosis	−0.7733	−0.3038
Shapiro–Wilk Test p	0.5175	0.0018

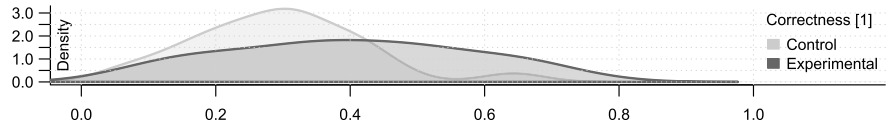

Fig. 15. Kernel density plot of *correctness* (ACS).

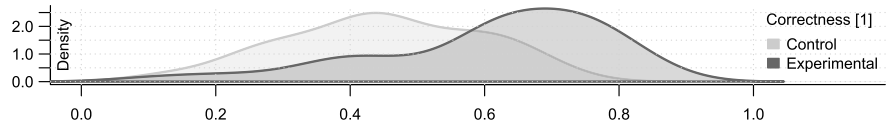

Fig. 16. Kernel density plot of *correctness* (TMD).

6.4 Descriptive Statistics

Correctness. Tables 4 and 5 show the number of corresponding observations, central tendency measures, and dispersion measures per group for the dependent variable *correctness*[14] These statistics are illustrated as a kernel density plot in Figures 15 and 16.

[14] *correctness* is defined as a value in $[0, 1] \cap \mathbb{R}$.

Table 6. Descriptive Statistics per Group of Dependent Variable *Duration* (ACS)

	Control	*Experimental*
Number of observations	47	47
Mean	2,025.49	1,673.68
Standard deviation	705.77	693.96
Median	1,980	1620
Median abs. deviation	622.69	680.51
Minimum	900	0
Maximum	4,320	3,457
Skew	1.0657	−0.1421
Kurtosis	1.3140	0.4589
Shapiro–Wilk Test p	0.0060	0.4023

Table 7. Descriptive Statistics per Group of Dependent Variable *Duration* (TMD)

	Control	*Experimental*
Number of observations	47	47
Mean	1,924.66	1,579.79
Standard deviation	990.65	732.48
Median	1,740	1,440
Median abs. deviation	745.75	444.78
Minimum	0	600
Maximum	5,280	4,620
Skew	1.3480	2.1688
Kurtosis	2.3119	5.9008
Shapiro–Wilk Test p	0.0003	0.0000

By visually inspecting the ACS correctness results (Figure 15) and examining the skew values in Table 4, it can be observed that the distribution of *Control* appears to be symmetrical to highly positively skewed, while the distribution of *Experimental* exhibits a lighter positive skew. The kurtosis values, which are less than 3 for both groups, indicate negative kurtosis. Moreover, for the TMD, visual inspection of Figure 16 and the skew values in Table 5 suggests that *Control* 's distribution is symmetrical to lightly negatively skewed. In contrast, *Experimental* 's distribution is highly negatively skewed.

Duration. Tables 6 and 7 show the number of observations, central tendency measures, and dispersion measures per group for the dependent variable *duration*[15]

Regarding duration results of ACS, the skews shown in Table 6 indicate that *Control* 's distribution is highly positively skewed, whereas *Experimental* 's distribution is moderately negatively skewed. The kurtosis values < 3 for both groups indicate negative kurtosis. Moreover, for TMD the skews in Table 7 indicate that both *Control* 's and *Experimental* 's distribution are highly positively skewed. The kurtosis value < 3 for *Control* group indicates negative kurtosis. However, for the *Experimental* group, the kurtosis value > 3 indicates highly tailed data.

6.5 Hypothesis Testing

Correctness and Duration. Suppose multiple groups are being compared using several dependent variables. In that case, it is customary to employ the **Multivariate Analysis of Variance (MANOVA)** statistical test under the condition that specific assumptions are satisfied [55]. This test helps determine whether independent variables impact the dependent variables, individually or in combination. As Section 6.4 mentions, the distribution of *Control* for *correctness* and *duration* does not significantly differ from the normal distribution. However, the distribution of *Experimental* significantly differs from the normal distribution for *correctness*, but not for *duration*. It is important to note that MANOVA requires all distributions to be normally distributed. Therefore, it was essential to compare the variances of both groups for both dependent variables to assess their equality, as this information would guide the selection of appropriate statistical tests. In this study, we opted to use Cliff's δ [56] as a robust and nonparametric test recommended by Kitchenham [35] for scenarios where data distribution, differing distributions between populations, or unequal variances are present. Although Cliff's δ was originally designed for measuring ordinal data, it is equally applicable to the quantitative and continuous data used in this study [57, 58]. This test estimates

[15] *Duration* is denoted in seconds.

Table 8. Hypothesis Tests per Group Combination of the Dependent Variable *Correctness* (ACS)

	Control vs. *Experimental*
Cliff's δ Test	
Cliff's δ	0.3459
s_δ	0.1137
v_δ	0.0129
z_δ	3.0416
CI_{low}	0.1065
CI_{high}	0.5473
$P(X > Y)$	0.3164
$P(X = Y)$	0.0213
$P(X < Y)$	0.6623
p	0.0031

Table 9. Hypothesis Tests per Group Combination of the Dependent Variable *Correctness* (TMD)

	Control vs. *Experimental*
Cliff's δ Test	
Cliff's δ	0.5505
s_δ	0.0984
v_δ	0.0097
z_δ	5.5916
CI_{low}	0.3293
CI_{high}	0.7144
$P(X > Y)$	0.2191
$P(X = Y)$	0.0113
$P(X < Y)$	0.7696
p	0.0000

the probability that a randomly selected observation from one group is larger than a randomly selected observation from another group, taking into account the reverse probability [59].

When conducting multiple hypothesis tests using a single method (in this case, Cliff's δ was applied twice), it is necessary to adjust the significance level (α) to mitigate the risk of Type I errors.[16] Several methods can be employed for α adjustment, such as the false discovery rate [60] or the Bonferroni-Dunn [61, 62] correction. The Bonferroni-Dunn correction is the most stringent form of correction and can be calculated using Equation (1):

$$\alpha' = \frac{\alpha}{n}, \tag{1}$$

where n is the number of times a test was applied. In our study, this results in $\alpha' = \frac{0.05}{2} = 0.025$ where $\alpha = 0.05$ and $n = 2$. The results of the one-tailed Cliff's δ test are shown in Tables 8 and 9 for *correctness* and Tables 10 and 11 for *duration*.

For *correctness*, Cliff's δ indicates by $p \leq \alpha'$ that *Experimental* scored significantly higher than *Control*. For *duration*, Cliff's δ yielded $p > \alpha'$, so we cannot conclude that *Experimental* took significantly longer than *Control* to complete the experiment. The negative Cliff's δ for *duration* indicates that, on average, the *Experimental* (which received additional IaC diagrams and metrics) completed the tasks in less time than the control group. This finding is interesting, as additional information might be expected to increase task duration due to the time needed to consult the supplementary materials, which is not the case here.

Based on Cliff's δ tests for both ACS and TMD, concerning *correctness*, we can reject the null hypothesis $H_0 1$ and thus accept the alternative hypothesis $H_a 1$. Conversely, with regard to *duration*, we are not able to reject the null hypothesis $H_0 2$ and reject the alternative hypothesis $H_a 2$.

Correlation Between Correctness and Duration. Upon visually examining the scatter plot in Figure 17, which explores potential correlations between the two dependent variables *correctness* and *duration*, no evident linear correlation was observed for either group. In the case of *Control*, there was a minimal increase in *correctness* concerning time. Similarly, for *Experimental*, although there seemed to be a rise in *correctness* with *duration*, the data points were widely dispersed from the indicated reference line, making it inappropriate to assume a linear correlation.

[16]It is important to note that there is no need to adjust the significance level (α) when conducting tests for normality or comparing variances.

Table 10. Hypothesis Tests per Group Combination of the Dependent Variable *Duration* (ACS)

	Control vs. *Experimental*
Cliff's δ Test	
Cliff's δ	−0.2499
s_δ	0.1146
v_δ	0.0131
z_δ	−2.1798
CI_{low}	−0.4588
CI_{high}	−0.0148
$P(X > Y)$	0.6129
$P(X = Y)$	0.0240
$P(X < Y)$	0.3631
p	0.0318

Table 11. Hypothesis Tests per Group Combination of the Dependent Variable *Duration* (TMD)

	Control vs. *Experimental*
Cliff's δ Test	
Cliff's δ	−0.2485
s_δ	0.1158
v_δ	0.0134
z_δ	−2.1462
CI_{low}	−0.4594
CI_{high}	−0.0111
$P(X > Y)$	0.6120
$P(X = Y)$	0.0244
$P(X < Y)$	0.3635
p	0.0345

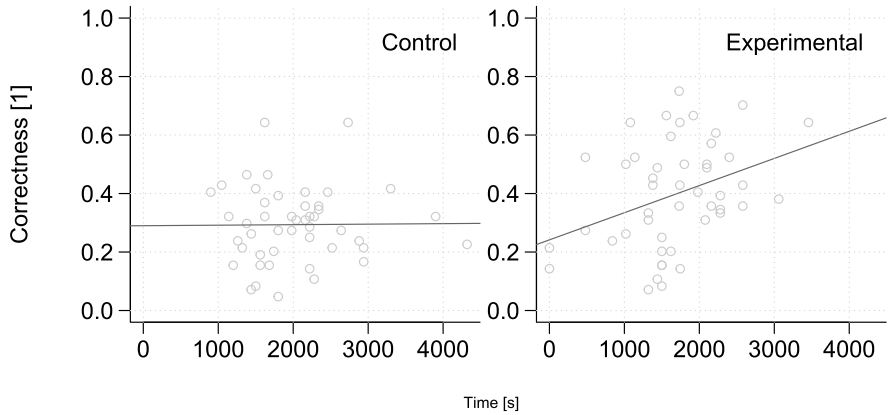

Fig. 17. Scatter plot per group of the dependent variables *correctness* to *duration* (ACS).

Likewise, for the TMD, a visual inspection of potential correlations in Figure 18 per group did not reveal any significant linear correlation. For *Control*, there was little to no increase in *correctness* over time. On the other hand, for *Experimental*, *correctness* appeared to decrease as time progressed.

After the visual inspection, we determined it necessary to perform a correlation test. Spearman's ρ test was selected for this purpose. For ACS, Spearman's ρ coefficients for *Control* (see Table 12) indicated a very weak positive association between correctness and duration. However, as indicated by $p > \alpha'$ (where α' is derived from the earlier adjustment of α), we are not able to reject the null hypothesis $H_0 3$ for both tests. For *Experimental*, the Spearman's ρ coefficients revealed a moderate positive association between *correctness* and *duration*. The p-value ($p > \alpha'$) indicates that the observed association is statistically significant, leading us to reject the null hypothesis $H_0 3$ and accept the alternative hypothesis $H_a 3$. Furthermore, the value of S obtained from Spearman's ρ test indicates that the ranks of the two variables under examination are not identical, providing additional evidence for a true association between the variables.

Regarding TMD, for *Control*, Spearman's ρ coefficients (see Table 13) also indicated a very weak positive association between *correctness* and *duration*. However, as explained earlier for ACS, we are not able to reject the null hypothesis $H_0 3$ and reject the alternative hypothesis $H_a 3$. For *Experimental*, Spearman's ρ coefficients yielded similar results to ACS, showing a moderate positive association

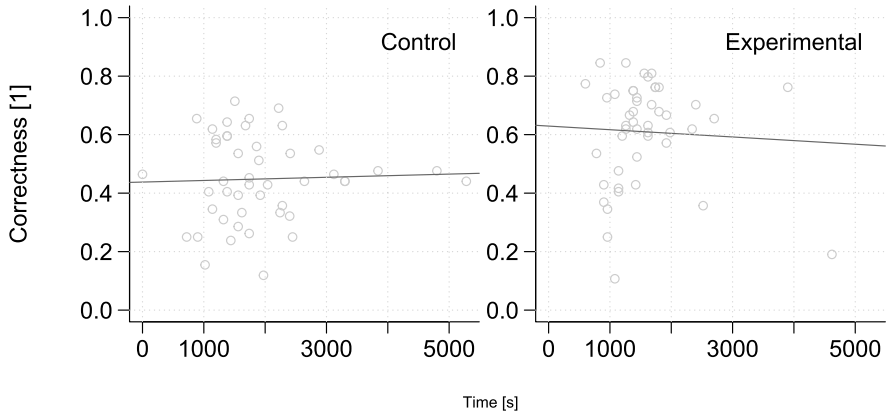

Fig. 18. Scatter plot per group of the dependent variables *correctness* to *duration* (TMD).

Table 12. Correlation per Group of the Dependent Variables *Correctness* with *Duration* per Group (ACS)

	Control	*Experimental*
Spearman's ρ	0.0016	0.3435
p	0.9917	0.0181
S	17,268.9319	11,354.1473

Table 13. Correlation per Group of the Dependent Variables *Correctness* with *Duration* per Group (TMD)

	Control	*Experimental*
Spearman's ρ	0.0383	0.1530
p	0.7984	0.3047
S	16,634.1006	14,650.4868

between *correctness* and *duration*. Therefore, we must also reject the null hypothesis H_03 and accept the alternative hypothesis H_a3.

6.6 Observation

Following the details provided in Section 4.5, participants were instructed to fill out a survey after each task to evaluate their confidence level in the accuracy of their responses. This self-assessment score was calculated using a formula that considers both the correctness (*correctness*) of participants' answers and their level of confidence in the correctness. The derived formula for calculating the self-assessment score is presented in Equation (2):

$$self_{assessment} = self_{correctness} - \frac{5 - self_{confidence}}{5}, \tag{2}$$

where $self_{correctness}$ represents the participants' average *correctness* as defined in 6.4, with 0 indicating entirely incorrect answers and 1 indicating completely correct answers.

The variable $self_{confidence} \in [0, 5] \cap \mathbb{R}$ represents the average confidence of participants based on a survey that utilized a five-point Likert scale. Each point on the scale is assigned equidistant values within the range of 0 to 5. A value of 0 signifies high confidence in the correctness of the answers, while a value of 5 indicates low confidence.

The variable $self_{assessment} \in [-1, 1] \cap \mathbb{R}$ represents the average self-assessment score of participants. A value less than 0 ($self_{assessment} < 0$) indicates that participants tend to overestimate the correctness of their answers. A value of 0 ($self_{assessment} = 0$) indicates that participants accurately estimate the correctness of their answers. A value greater than 0 ($self_{assessment} > 0$) indicates that participants tend to underestimate the correctness of their answers.

Figure 19 illustrates the kernel density plot of participants' overall self-assessment score per group. The plot shows that, on average, participants in the *Control* group accurately estimated their

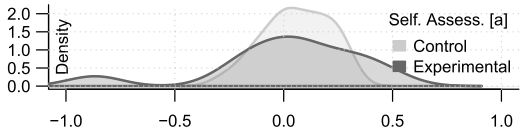

Fig. 19. Kernel density plot per group of participants' self assessment (ACS).

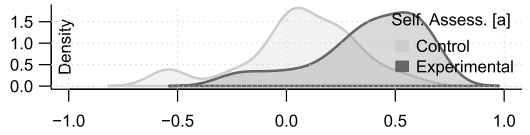

Fig. 20. Kernel density plot per group of participants' self assessment (TMD).

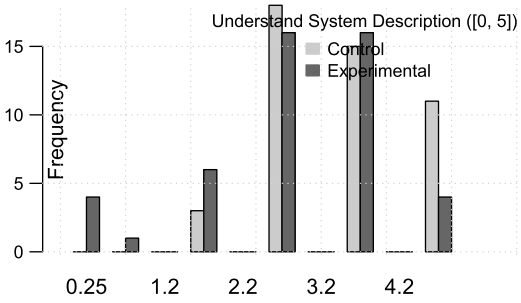

Fig. 21. Histogram per group of participants' understanding of system descriptions (ACS).

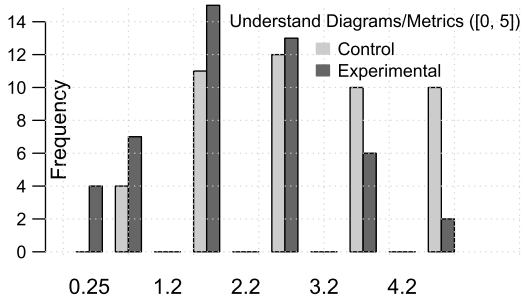

Fig. 22. Histogram per group of participants' understanding of semi-formal models and metrics (ACS).

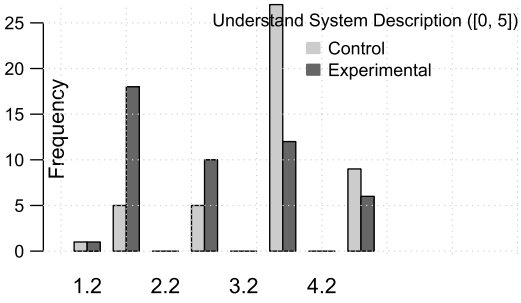

Fig. 23. Histogram per group of participants' understanding of system descriptions (TMD).

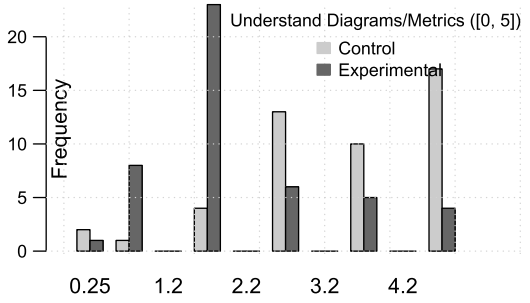

Fig. 24. Histogram per group of participants' understanding of semi-formal models and metrics (TMD).

correctness across all tasks, while participants in the *Experimental* group slightly underestimated their *correctness*.

Figure 20 indicates that *Control* slightly underestimated their *correctness* on average across all tasks, whereas *Experimental* underestimated their *correctness*.

Figures 21 and 22 visually represent the distribution of understanding scores for semi-formal models, metrics, and system descriptions by participants in the *Control* and *Experimental*. These histograms showcase the frequency of scores across a range from 0 to 5, with clear distinctions between the two groups.

Figures 23 and 24 visually complement the statistical data, illustrating the distribution of understanding scores for system descriptions across a range from 0 to 5. The *Control* 's distribution is more concentrated towards higher scores (around 3.2 to 4.2), aligning with the higher mean and median values noted. This visualization supports the statistical finding of better system description comprehension in the *Control*.

7 Discussion

7.1 Interpretation of the Results

Correctness and Duration. In both ACS and TMD, when testing $H_0 1$, we observed a significant difference in task *correctness*, indicating that providing formal IaC system semi-formal models and metrics alongside source code is more effective. Similarly, when testing $H_0 2$, we found no significant difference in task *duration* between *Control* and *Experimental*. Regarding $H_0 3$, we discovered that for *Control*, there was no significant positive correlation between *correctness* and *duration*. However, for *Experimental*, we found a significant positive correlation between *correctness* and *duration*.

Our findings provide strong evidence in favor of the notion that the inclusion of semi-formal models and metrics greatly improves the comprehension of IaC security practices, as measured by *correctness*. The integration of these supplementary resources alongside the source code proves advantageous when engaging practitioners in understanding IaC security practices and executing associated tasks. These outcomes align consistently with our initial expectations.

After participants in the *Experimental* group read and comprehended the material, their task completion time did not show a notable increase compared to that of those in the *Control* group despite having access to a greater amount of reference material. One possible explanation for this absence of a significant difference in *duration* is that participants in the *Experimental* group primarily relied on the semi-formal IaC system models and metrics to locate the relevant information during task completion. They were content with finding it relatively quickly. In contrast, participants in the *Control* group were limited in their search for details in the source code. Assuming that *duration* is inversely linked to understanding, it can be inferred that providing semi-formal IaC system models and metrics as supplementary material does not impede comprehension. If participants in the *Experimental* group had taken significantly longer to complete the tasks, regardless of their *correctness*, it could have indicated that semi-formal IaC system models and metrics may pose a barrier or increase the learning curve for understanding IaC system descriptions to some extent. These results are generalized to both students and developers who have a similar level of familiarity with understanding the IaC system, as mentioned previously in Section 4.4.

Correlation Between Correctness and Duration. Our initial assumption and intuition led us to believe that there would be a strong correlation between *correctness* and *duration*. However, contrary to what we anticipated, there was no significant positive correlation found between *correctness* and *duration* in the context of *Control*. This indicates that participants in *Control* were satisfied with achieving accurate answers within a reasonable timeframe and proceeded to the next task accordingly. However, for ACS, in Figure 17, it can be seen that clearly *Experimental* improves with more time more correctness, whereas this is not visible in *Control*.

In contrast to our initial assumptions, our findings in the case of *Experimental* revealed a significant positive correlation between *correctness* and *duration*. It is possible that participants in *Experimental* adopted a distinct approach to answering the questions. Since the only difference between *Control* and *Experimental* was the inclusion of semi-formal IaC system models and metrics, it is plausible that participants heavily relied on these resources in *Experimental*. Another possibility is that participants first consulted the source code for answers and then utilized the semi-formal IaC system models and metrics to review and amend their responses, leading to the observed positive correlation between *correctness* and *duration*.

Therefore, practitioners who engage in tasks based on source code should be aware that spending excessive time on such tasks does not necessarily result in better performance in terms of *correctness*. Participants in *Control* took a similar amount of time on average to those in the *Experimental* group before being satisfied with their answers. This suggests that other factors may come into play. In this study, providing semi-formal IaC system models and metrics is crucial when time constraints

are equal. This result further strengthens the case for providing practitioners with semi-formal IaC system models and metrics. Similar to the previous correlation, these findings are also generalized to students and developers who share a comparable understanding.

Self Assessment. In Section 6.6, our analysis of participants' survey responses for ACS revealed an interesting finding: participants who did not receive semi-formal IaC system models and metrics correctly estimated their performance. In contrast, the opposite was observed for those provided with these supplementary materials. In the TMD study, participants who did not receive semi-formal IaC system models and metrics slightly underestimated their performance, as did those who received them. Providing additional semi-formal IaC system models and metrics may have diminished the confidence. It is possible that these participants felt overwhelmed by the extra information or doubted their complete comprehension of the semi-formal IaC system models and metrics. This outcome contradicts our initial expectations, as we anticipated higher confidence levels among participants who performed better. This finding neither strongly supports nor opposes the utilization of semi-formal IaC system models and metrics. The preference of practitioners, whether to be overconfident, underconfident, or somewhere in between, is subjective and reliant on the context, and it cannot be solely inferred from the relationship between *correctness* and *duration*. Consequently, this observation remains neutral, lacking a discernible positive or negative influence on the hypothesis tests or our conclusions. Nevertheless, we acknowledge its importance as a factor to consider attentively throughout our analysis.

Understanding of Experiment Materials. From Figures 21 and 22 in Section 6.6, it's evident that the *Control* generally had higher scores, as indicated by the distribution leaning towards the 3.2 to 4.2 range. This visual representation corroborates the statistical data that showed a higher average understanding in the *Control* compared to the *Experimental*. The *Experimental*, on the other hand, shows a wider spread of scores, starting from 0, which indicates a broader range of understanding levels among its participants, including some who did not grasp the semi-formal models at all.

The analysis of participants' understanding of text content, as illustrated in Figures 23 and 24 in Section 6.6, reveals notable differences between the *Control* and *Experimental*. The *Control* demonstrated superior comprehension, with a higher mean understanding compared to the *Experimental*. The histograms show a concentration of higher scores within the *Control*, predominantly around the 3.2 to 4.2 range, indicating a stronger grasp of the system description content.

The overarching theme from both datasets is that the *Control* consistently demonstrated a higher and more uniform understanding of semi-formal models, metrics, and system description content. This suggests that the *Control* were more effective in facilitating comprehension of complex information. The *Experimental*, while showing a broad range of understanding, tended to have more participants struggling with both description, semi-formal models and metrics, as indicated by lower mean scores and a broader range of scores.

These results could imply that the provided materials might have been less effective or that the *Experimental* had other barriers to understanding. Both the *Control* and *Experimental* showed better average understanding for system description content than for semi-formal models and metrics, with the *Control* outperforming the *Experimental* in both domains. This pattern might suggest a positive correlation within each group, where individuals who understood semi-formal models and metrics well also tended to understand system descriptions well.

8 Related Work

This section provides an overview of the existing literature on IaC best practices, frameworks, and metrics, along with studies that employ comparable methodologies to our research.

As IaC practices are becoming increasingly popular and widely adopted by the industry, more scientific research is being conducted into collecting and systematizing IaC-related patterns and

practices. Kumara et al. [10] present a broad catalog of best and bad practices, both language-agnostic and language-specific, that reflect implementation issues, design issues, and violations of essential IaC principles. Morris [1] presents a collection of guidances on managing Infrastructure-as-Code. In his book, there is a detailed description of technologies related to IaC-based practices and a broad catalog of patterns and practices embodied in several categories. Language-specific practices have been proposed by Sharma et al. [14], who present a catalog of design and implementation smells for Puppet. Schwarz et al. [15] present a catalog of smells for Chef. Our work also follows IaC-specific recommendations described by AWS [16], OWASP [17–19], and the Cloud Security Alliance [20, 21]. Unlike our research, several of the mentioned studies have a narrower focus on implementation issues or decisions in the deployment architecture of the systems being deployed. They do not delve into the specific aspects of security checking, which our work emphasizes.

A few studies investigate security smells in IaC scripts [9–11]. In contrast to our work, none of these work investigates architectural security issues but focus on implementation-level issues.

Despite its many advantages, the transition to IaC introduces challenges, with state reconciliation standing out as a critical concern. State reconciliation is the process of ensuring that the actual state of the infrastructure aligns with the as-coded desired state [63, 64]. Discrepancies between these two states can lead to operational inefficiencies and heightened security risks. Maintaining congruence between the desired and actual states is essential for the operational integrity of IaC practices, underscoring the importance of observability and continuous monitoring in securing IaC environments. Note that our work focuses only on those aspects visible in the deployment model; other aspects of state reconciliation would need runtime data to be studied adequately, which is out of the scope of our work.

Several studies propose tools and metrics for assessing and improving the quality of IaC deployment models. Dalla Palma et al. [65, 66] propose a catalog of 46 quality metrics focusing on Ansible scripts to identify IaC-related properties and show how to use them in analyzing IaC scripts. Wurster et al. [67] present **Topology and Orchestration Specification for Cloud Applications (TOSCA)** Lightning, an integrated toolchain for specifying multi-service applications with TOSCA Light and transforming them into different deployment technologies. They also present a case study on the toolchain's effectiveness based on a third-party application and Kubernetes. A tool-based approach for detecting smells in TOSCA models is proposed by Kumara et al. [68]. Sotiropoulos et al. [69] developed a tool-based approach that identifies dependencies-related issues by analyzing Puppet manifests and their system call trace. Van der Bent et al. [70] define metrics reflecting the best practices to assess Puppet code quality. While these works provide valuable insights into enhancing the quality and analysis of IaC deployment models, our research extends beyond them by focusing specifically on the intersection of security practices and metrics within the context of IaC architectures.

Pendleton et al. [71] present a comprehensive survey on security metrics. It focuses on how a system security state can evolve due to cyber-attack defense interactions. They then propose a security metrics framework for measuring system-level security. While certain works concentrate on quality assurance in IaC systems, none specifically tackle the crucial aspect of security concerns and measures in IaC deployment models, which is the primary focus of our research.

Heijstek et al. [72] conducted a controlled experiment that shares similarities with our study. They aimed to investigate the effectiveness of visual versus textual artifacts in communicating software design decisions to software developers. Their research recruited forty-seven participants from industry and academia, who evaluated both UML representations as diagrammatic artifacts and informal textual descriptions. However, there are notable differences between their study and ours. Firstly, all participants assessed both representations, whereas in our research, there may be variations in the evaluation process. Secondly, our study focuses specifically on security practices in IaC, which distinguishes it from the broader scope of their investigation. These methodological

dissimilarities highlight the unique aspects of our study and contribute to the broader body of research in this field.

Allodi, Cremonini et al. [73] conducted a study involving seventy-three participants to evaluate the accuracy of security professionals and students with advanced technical education in assessing the severity of software vulnerabilities based on various attributes. Unlike our focus on comparing different system description methods, they emphasized participants' background knowledge and education. A notable difference in methodology is the classification of participants into three groups: students with a BSc in information security enrolled in an MSc in information security degree program, students in an MSc in computer science program, and security practitioners. Additionally, they specifically recruited students with no professional expertise, distinguishing their approach from ours.

Allodi, Biagioni et al. [74] conducted a controlled experiment involving twenty-nine students to investigate the challenges participants face in assessing system vulnerabilities when security requirements change. In contrast to our study, they employed a within-subject design, focusing on variations in system requirements rather than modeling differences. However, like our approach, they formulated hypotheses and subjected them to testing using statistical methods.

Labunets et al. [75] conducted a controlled experiment involving twenty-nine MSc students to explore the participants' perceptions of visual and textual methods for security risk assessment in terms of effectiveness. Although their study shares similarities with ours in comparing visual and textual representations, there are notable differences. Firstly, the research does not explicitly examine the distinctions between formal and informal representations, nor does it around system understanding, which are key focuses of our study. Additionally, their experiment addresses security aspects but not in the IaC domain, our research's primary area of interest. These differences highlight our study's specific scope and objectives while acknowledging the valuable insights provided by Labunets et al.'s experiment in the broader context of comparing visual and textual methods for assessment.

Verdet et al. [22] analyzed how practitioners tackle security challenges through IaC. They identify and assess Terraform security practices for major cloud providers across 812 open-source projects on GitHub. Access policies are widely adopted, while Encryption in rest policies are often overlooked. They also find a strong correlation between GitHub stars and adoption of best practices. Their findings provide guidance for cloud practitioners to enhance security measures. While this study offers useful insights, our work goes further by concentrating specifically on particular security practices and metrics applicable to both Ansible and Terraform.

While the studies provide valuable insights into security practices, our research delves deeper, focusing specifically on the intricacies of security practices and metrics within the domains of Ansible and Terraform. By zooming in on these two widely utilized IaC tools, our aim was to unearth and assess the comprehension of these technologies. Through this concentrated analysis, we aim to contribute to a more nuanced understanding of how semi-formal models and metrics can support in understanding specific security practices in the IaC domain.

9 Threats to Validity

Threats to Internal Validity. The experimental sessions had no disruptions or incidents that interfered with the process. Participants received an introduction and were allowed to address any queries they had. No questions that broadly impacted the sessions emerged; instead, individual participant questions were addressed individually.

The limited time allocated to each session effectively minimized the potential for maturation effects; indeed, no such effects were observed. The experimental design ensured that each participant's contribution to the experiment results occurred only once, eliminating the possibility of

learning between sessions. Therefore, any learning effect within a session across the tasks does not favor either *Control* or *Experimental*. Additionally, each participant had an equal opportunity to score points for their performance, irrespective of their assigned group, eliminating instrumental bias. Furthermore, the random assignment of participants to groups prevented any selection bias.

While preventing participants from discussing the experiment with future participants is impossible, measures were taken to minimize the potential for cross-contamination between experimental sessions and groups. Participants were not allowed to take a copy of the experiment sheet with them or use electronic devices during the experiment. The complexity of the systems and tasks and the consecutive spacing of the sessions further reduced the likelihood of participants gaining unfair knowledge before their sessions. Additionally, randomly assigning participants to groups ensured that any advantage would be evenly distributed without favoring either group. The prohibition of electronic devices also prevented participants from accessing external information sources, as outlined in Section 5.3. The only permissible reference materials were the printed information document described in Section 4.5 and the limited access to the source code, thereby preventing potential effects of participants consulting other sources.

Threats to External Validity. A potential threat to external validity is the sample size of our study, which may be insufficient to yield statistically significant results. To mitigate this risk, we have employed robust statistical methods suitable for our sample size, ensuring the analysis is conducted appropriately and accurately given the available data.

Another external validity concern arises from using students instead of non-student professionals in our study, raising questions about the generalizability of our findings to practical settings. To address this, we tried to familiarize the students with the IaC security-related concepts employed in the experiment. Notably, all participants possessed diverse theoretical knowledge in software engineering, distributed systems, programming experience, and industry exposure.

While we acknowledge the potential limitations of a student sample and potential threats to the generalization of students and developers with similar backgrounds, it's important to note that the characteristics of our student population may align with those of a broader community of software developers. In the Stack Overflow industry survey (see Section 4.4), 69% of respondents were self-taught, 43% held a bachelor's degree in computer science or a related field, 19% had a master's degree, and 2% had a PhD. These statistics reveal that a significant portion of the fifty thousand developers surveyed, even on a widely respected software development platform, are self-taught and lack formal degrees. This suggests that a degree may not be a prerequisite for qualifying as a professional developer.

Given this context, our assertion is strengthened that students can reasonably stand in as substitutes for developers in our study. Consequently, we propose that our findings may apply to professional software developers, at least to some extent. However, it would be prudent to replicate similar experiments with practitioners to confirm the absence of significant differences compared to the population of professional developers.

Our example systems are of modest size, and while we ensured they contain realistically implemented best practices (comparing to our studies of gray literature and 21 open source systems [24]), it cannot be guaranteed that the same results would be achievable in large-scale industrial systems. We opted for the smaller systems to avoid fatigue effects and other such biases. However, as in a larger, more realistic system, it would be harder to find the relevant IaC scripts, it is likely that in real-life settings, the control group that only used the source code would be at an even greater disadvantage.

Threats to Construct Validity. Construct validity threats arise from uncertainties about whether the measurement and operationalization of variables accurately represent the theoretical constructs being studied. These threats involve concerns about the suitability of chosen measurement methods,

including issues like unclear operational definitions, insufficiently sensitive instrumentation, or potential biases introduced by the experimenter.

In Section 4.6, we focused on *correctness* and *duration* as dependent variables for measuring understandability, but we acknowledge the possibility that other metrics might be more appropriate. Additionally, there could be more suitable methods for gauging participants' confidence in addressing our research question.

When participants self-record times, ensuring that these measurements accurately represent task completion times is crucial. Potential threats to construct validity in this context include ambiguous or imprecise definitions of task completion, leading to inconsistent timekeeping among participants. However, we aimed to mitigate this risk by providing clear instructions and guidance.

To address the threat of self-timing reliability, we rigorously assessed the reliability of each recorded timestamp during dataset preparation, excluding inconsistent or unrealistic recordings. We also considered missing or implausible timestamp values, minimizing the risk of unreliable timekeeping affecting our conclusions.

Participants were instructed to use a common, centrally controlled clock with high readability and accuracy to mitigate erroneous measurement accuracy. This eliminated potential issues such as manual timer manipulation, misreading due to poor readability, or using different clocks, enhancing the validity of our timekeeping data.

Threats to Content Validity. We cannot conceive of any threats to content validity since the experiment topic and subject matter were relevant to all participants' university courses, regardless of group assignment. Furthermore, the information sheet provided to participants provided sufficient prerequisite knowledge to participate in the study. Any inconsistencies or ambiguities in the experiment material would have affected both groups equally.

Threats to Conclusion Validity. Given that the experiment topic and subject matter were relevant to all participants' university courses, irrespective of their group assignment, we do not foresee any threats to content validity. Moreover, the information sheet provided to participants offered the necessary prerequisite knowledge for their participation in the study. Any inconsistencies or ambiguities in the experiment material would have affected both groups equally, ensuring fairness.

10 Conclusion

The results of our study provide strong support for the effectiveness of incorporating semi-formal IaC system models and metrics in enhancing the understanding of IaC security practices, as measured by *correctness*. Including these additional resources alongside source code proved beneficial in aiding practitioners in comprehending IaC security practices and performing related tasks.

Interestingly, despite participants in the *Experimental* group having access to more reference material, their task completion time (*duration*) did not significantly increase compared to the *Control* group. This suggests that participants in the *Experimental* group primarily relied on the semi-formalIaC system models and metrics to quickly locate the relevant information without impeding their comprehension. In contrast, participants in the *Control* group had only access to source code, which may have influenced their task completion time.

Contrary to our initial expectations, there was no significant positive correlation between *correctness* and *duration* in the *Control* group. This indicates that participants in *Control* were satisfied with finding correct answers within a reasonable time frame and proceeded to the next task. However, in the case of *Experimental*, there was a significant positive correlation between *correctness* and *duration*. This suggests that participants relied on the semi-formal IaC system models and metrics to review and correct their responses after checking the source code.

Practitioners should be aware that spending excessive time on tasks based solely on source code does not necessarily lead to better performance in terms of *correctness*. In our study, semi-formal IaC

system models and metrics emerged as a crucial factor in achieving higher *correctness*, regardless of the time spent on the tasks. This finding further supports the case for providing practitioners with semi-formal IaC system models and metrics to enhance their understanding and performance.

An interesting finding emerged when analyzing participants' survey responses. Participants who did not receive semi-formal IaC system models and metrics accurately estimated their performance, while those who had access to these resources underestimated their performance. This observation suggests that additional semi-formal IaC system models and metrics may have diminished participants' confidence due to feeling overwhelmed or uncertain about their full comprehension of the materials. While contradicting our initial expectations, this outcome does not strongly support or oppose using semi-formal IaC system models and metrics, as confidence levels are subjective and context-dependent.

Acknowledgement

This research was funded in whole or in part by the Austrian Science Fund (FWF) project "Infrastructure-as-code Architecture Decision Compliance (IAC2)," project number: I 4731. For open access purposes, the author has applied a CC BY public copyright license to any author accepted manuscript version arising from this submission.

References

[1] K. Morris. 2020. *Infrastructure as Code: Dynamic Systems for the Cloud*. Vol. 2. O'Reilly.

[2] J. Humble and D. Farley. 2010. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Addison-Wesley Professional.

[3] M. Nygard. 2007. *Release It! Design and Deploy Production-Ready Software*. Pragmatic Bookshelf.

[4] M. Artac, T. Borovssak, E. Di Nitto, M. Guerriero, and D.A. Tamburri. 2017. DevOps: Introducing infrastructure-as-code. In *Proceedings of the IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, 497–498.

[5] E. Özdoğan, O. Ceran, and M. T. Üstündağ. 2023. Systematic analysis of infrastructure as code technologies. *Gazi University Journal of Science Part A: Engineering and Innovation* 10, 4 (2023), 452–471. DOI: https://doi.org/10.54287/gujsa.1373305

[6] S. Sengupta, V. Kaulgud, and V. S. Sharma. 2011. Cloud computing security–trends and research directions. In *Proceedings of the IEEE World Congress on Services*. IEEE, 524–531.

[7] M. Guerriero, M. Garriga, D. A. Tamburri, and F. Palomba. 2019. Adoption, support, and challenges of infrastructure-as-code: Insights from industry. In *Proceedings of the IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 580–589.

[8] A.-C. Iosif, T. E. Gasiba, T. Zhao, U. Lechner, and M. Pinto-Albuquerque. 2021. A large-scale study on the security vulnerabilities of cloud deployments. In *Proceedings of the Inernational Conference on Ubiquitous Security*. Springer, 171–188.

[9] A. Rahman, C. Parnin, and L. Williams. 2019. The seven sins: Security smells in infrastructure as code scripts. In *Proceedings of the IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, 164–175. DOI: https://doi.org/10.1109/ICSE.2019.00033

[10] I. Kumara, M. Garriga, A. U. Romeu, D. DiNucci, F. Palomba, D. A. Tamburri, and W.-J. van den Heuvel. 2021. The do's and don'ts of infrastructure code: A systematic gray literature review. *Information and Software Technology* 137 (2021), 106593.

[11] A. Rahman, M. R. Rahman, C. Parnin, and L. Williams. 2021. Security smells in ansible and chef scripts: A replication study. *ACM Transactions on Software Engineering and Methodology* 30, 1 (2021), 1–31.

[12] IEEE Computer Society. 2004. Avoiding the Top 10 Software Security De-Sign Flaws. Retrieved from https://ieeecs-media.computer.org/media/technical-activities/CYBSI/docs/Top-10-Flaws.pdf

[13] OWASP. 2021. Top 10 Vulnerabilities. Retrieved from https://owasp.org/www-project-top-ten/

[14] T. Sharma, M. Fragkoulis, and D. Spinellis. 2016. Does your configuration code smell?. In *Proceedings of the 13th International Conference on Mining Software Repositories (MSR '16)*. ACM, New York, NY, 189–200.

[15] J. Schwarz, A. Steffens, and H. Lichter. 2018. Code smells in infrastructure as code. In *Proceedings of the 11th International Conference on the Quality of Information and Communications Technology (QUATIC)*, 220–228. DOI: https://doi.org/10.1109/QUATIC.2018.00040

[16] AWS Documentation. 2021. Security Groups for Your VPC. Retrieved from https://docs.aws.amazon.com/vpc/latest/userguide/VPC_SecurityGroups.html
[17] OWASP Cheat Sheet Series. 2021. Authentication Cheat Sheet. Retrieved from https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html#logging-and-monitoring
[18] OWASP Cheat Sheet Series. 2021. Transport Layer Protection Cheat Sheett. Retrieved from https://cheatsheetseries.owasp.org/cheatsheets/Transport_Layer_Protection_Cheat_Sheet.html#ssl-vs-tls
[19] OWASP Cheat Sheet Series. 2021. Infrastructure as Code Security Cheatsheet. Retrieved from https://cheatsheetseries.owasp.org/cheatsheets/Infrastructure_as_Code_Security_Cheat_Sheet.html
[20] Cloud Security Alliance. 2018. Continuous Monitoring in the Cloud. Retrieved from https://cloudsecurityalliance.org/blog/2018/06/11/continuous-monitoring-in-the-cloud/
[21] Cloud Security Alliance. 2021. Five Approaches for Securing Identity in Cloud Infrastructure. Retrieved from https://cloudsecurityalliance.org/blog/2021/05/20/five-approaches-for-securing-identity-in-cloud-infrastructure/
[22] A. Verdet, M. Hamdaqa, L. Da Silva, and F. Khomh. 2023. Exploring security practices in infrastructure as code: An empirical study. arXiv:2308.03952. Retrieved from https://arxiv.org/abs/2308.03952
[23] G. Gurbatov. 2022. *A Comparison between Terraform and Ansible on Their Impact Upon the Lifecycle and Security Management for Modifiable Cloud Infrastructures in OpenStack*. Master Thesis. Faculty of Computing, Blekinge Institute of Technology, Sweden.
[24] E. Ntentos, U. Zdun, G. Falazi, U. Breitenbücher, and F. Leymann. 2022. Assessing architecture conformance to security-related practices in infrastructure as code based deployments. In *Proceedings of the IEEE International Conference on Services Computing (IEEESCC '22)*, 123–133. DOI: https://doi.org/10.1109/SCC55611.2022.00029
[25] Okta. 2021. Token-Based Authentication. Retrieved from https://www.okta.com/identity-101/what-is-token-based-authentication/
[26] Google Cloud. 2021. Using API Keys. Retrieved from https://cloud.google.com/docs/authentication/api-keys
[27] auth0Docs. 2021. Single Sign-On (SSO). Retrieved from https://auth0.com/docs/authenticate/single-sign-on
[28] The Security Skeptic. 2021. Firewall Best Practices – Egress Traffic Filtering. Retrieved from https://securityskeptic.typepad.com/the-security-skeptic/firewall-best-practices-egress-traffic-filtering.html
[29] kubernetes Documentation. 2021. Ingress Traffic Control. Retrieved from https://kubernetes.io/docs/concepts/services-networking/ingress/
[30] U. Zdun, E. Navarro, and F. Leymann. 2017. Ensuring and assessing architecture conformance to microservice decomposition patterns. In *Service-Oriented Computing*. M. Maximilien, A. Vallecillo, J. Wang, and M. Oriol (Eds.), Springer International Publishing, Cham, 411–429.
[31] A. Jedlitschka, M. Ciolkowski, and D. Pfahl. 2008. Reporting experiments in software engineering. In *Guide to Advanced Empirical Software Engineering*. J. Singer, F. Shull, and D. Sjøberg (Eds.), Springer. 201–228. DOI: https://doi.org/10.1007/978-1-84800-044-5_8
[32] B. Kitchenham, S. L. Pfleeger, L. M. Pickard, P. Jones, D. Hoaglin, K. Emam, and J. Rosenberg. 2002. Preliminary guidelines for empirical research in software engineering. *IEEE Transactions on Software Engineering* 28 (2002), 721–734. DOI: https://doi.org/10.1109/TSE.2002.1027796
[33] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén. 2012. *Experimentation in Software Engineering*. Springer. DOI: https://doi.org/10.1007/978-3-642-29044-2
[34] N. Juristo and A. Moreno. 2001. *Basics of Software Engineering Experimentation*. Springer. DOI: https://doi.org/10.1007/978-1-4757-3304-4
[35] B. Kitchenham, L. Madeyski, D. Budgen, J. Keung, P. Brereton, S. Charters, S. Gibbs, and A. Pohthong. 2016. Robust statistical methods for empirical software engineering. *Empirical Software Engineering* 22, 2 (Apr. 2017), 579–630. DOI: https://doi.org/10.1007/s10664-016-9437-5
[36] G. Charness, U. Gneezy, and M. A. Kuhn. 2012. Experimental methods: Between-subject and within-subject design. *Journal of Economic Behavior & Organization* 81, 1 (2012), 1–8.
[37] M. Höst, B. Regnell, and C. Wohlin. 2000. Using students as subjects – A comparative study of students and professionals in lead-time impact assessment. *Empirical Software Engineering* 5 (2000), 201–214. DOI: https://doi.org/10.1023/A:1026586415054
[38] P. Runeson. 2003. Using students as experiment subjects – An analysis on graduate and freshmen student data. In *Proceedings of the 7th International Conference on Empirical Assessment & Evaluation in Software Engineering*, 95–102.
[39] M. Svahnberg, A. Aurum, and C. Wohlin. 2008. Using students as subjects – an empirical evaluation. In *Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '08)*. ACM, New York, NY, 288–290. DOI: https://doi.org/10.1145/1414004.1414055
[40] I. Salman, A. T. Misirli, and N. J. Juzgado. 2015. Are students representatives of professionals in software engineering experiments? In *Proceedings of the IEEE/ACM 37th IEEE International Conference on Software Engineering* 1 (2015), 666–676.

[41] D. Falessi, N. Juristo, C. Wohlin, B. Turhan, J. Münch, A. Jedlitschka, and M. Oivo. 2018. Empirical software engineering experts on the use of students and professionals in experiments. *Empirical Software Engineering* 23 (2018), 452–489. DOI: https://doi.org/10.1007/s10664-017-9523-3

[42] J. Siegmund, C. Kästner, S. Apel, J. Liebig, M. Schulze, R. Dachselt, M. Papendieck, T. Leich, and G. Saake. 2012. Do background colors improve program comprehension in the #ifdef hell? *Empirical Software Engineering* 18 (2012), 1–47. DOI: https://doi.org/10.1007/s10664-012-9208-x

[43] B. Hoisl, S. Sobernig, and M. Strembeck. 2014. Comparing three notations for defining scenario-based model tests: A controlled experiment. *Proceedings of the 9th International Conference on the Quality of Information and Communications Technology (QUATIC '14)*, 95–104. DOI: https://doi.org/10.1109/QUATIC.2014.19

[44] C. Czepa and U. Zdun. 2019. How understandable are pattern-based behavioral constraints for novice software designers? *ACM Transactions on Software Engineering and Methodology* 28 (2019), 1–38.

[45] C. Czepa and U. Zdun. 2020. On the understandability of temporal properties formalized in linear temporal logic, property specification patterns and event processing language. *IEEE Transactions on Software Engineering* 46 (2020), 100–112.

[46] P. Paulweber, G. Simhandl, and U. Zdun. 2021. On the understandability of language constructs to structure the state and behavior in abstract state machine specifications: A controlled experiment. *Journal of Systems and Software* 178 (2021), 110987.

[47] M. Vujovic, D. Hernández-Leo, R. Martinez-Maldonado, M. Cukurova, and D. Spikol. 2022. Multimodal learning analytics and the design of learning spaces. In *The Multimodal Learning Analytics Handbook*. Springer, 31–49.

[48] J. Wong, M. Baars, D. Davis, T. Van Der Zee, G.-J. Houben, and F. Paas. 2019. Supporting self-regulated learning in online learning environments and MOOCs: A systematic review. *International Journal of Human–Computer Interaction* 35, 4–5 (2019), 356–373.

[49] K. Xie, B. C. Heddy, and B. A. Greene. 2019. Affordances of using mobile technology to support experience-sampling method in examining college students' engagement. *Computers & Education* 128 (2019), 183–198.

[50] S. S. Shapiro and M. B. Wilk. 1965. An analysis of variance test for normality (complete samples). *Biometrika* 52 (1965), 591–611.

[51] N. M. Razali and B. Yap. 2011. Power comparisons of Shapiro-Wilk, Kolmogorov-Smirnov, Lilliefors and Anderson-Darling tests. *Journal of Statistical Modeling & Analytics* 2 (01 2011).

[52] T. W. Anderson and D. A. Darling. 1954. A test of goodness of fit. *The Journal of the American Statistical Association* 49, 268 (1954), 765–769. DOI: https://doi.org/10.1080/01621459.1954.10501232

[53] H. W. Lilliefors. 1967. On the Kolmogorov-Smirnov test for normality with mean and variance unknown. *The Journal of the American Statistical Association* 62 (1967), 399–402.

[54] A. N. Kolmogorov. 1933. Sulla determinazione empirica di una legge didistribuzione. *Giorn Dell'inst Ital Degli Att* 4 (1933), 89–91.

[55] J. H. Bray and S. E. Maxwell. 1982. Analyzing and interpreting significant MANOVAs. *Review of Educational Research* 52, 3 (1982), 340–367. Retrieved from http://www.jstor.org/stable/1170422

[56] N. Cliff. 1993. Dominance statistics: Ordinal analyses to answer ordinal questions. *Psychological Bulletin* 114 (1993), 494–509.

[57] H. D. Delaney and A. Vargha. 2002. Comparing several robust tests of stochastic equality with ordinally scaled variables and small to moderate sized samples. *Psychological Methods* 7 4 (2002), 485–503.

[58] L. M. Hsu. 2004. Biases of success rate differences shown in binomial effect size displays. *Psychological Methods* 9, 2 (2004), 183–197.

[59] N. Cliff. 2010. Answering ordinal questions with ordinal data using ordinal statistics. *Multivariate Behavioral Research* 31 (2010), 331–350. DOI: https://doi.org/10.1207/s15327906mbr3103_4

[60] Y. Benjamini and Y. Hochberg. 1995. Controlling the false discovery rate – A practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society Series B* 57 (1995), 289 – 300. DOI: https://doi.org/10.2307/2346101

[61] C. Bonferroni. 1936. Teoria statistica delle classi e calcolo delle probabilita. *Pubblicazioni del R Istituto Superiore di Scienze Economiche e Commerciali di Firenze* 8 (1936), 3–62.

[62] O. J. Dunn. 1961. Multiple comparisons among means. *The Journal of the American Statistical Association* 56 (1961), 52–64.

[63] A. Rahman and C. Parnin. 2023. Detecting and characterizing propagation of security weaknesses in puppet-based infrastructure management. *IEEE Transactions on Software Engineering* 49, 6 (Jun. 2023), 3536–3553. DOI: https://doi.org/10.1109/TSE.2023.3265962

[64] X. Sun, W. Luo, J. Tyler, A. Ganesan, R. Alagappan, M. Gasch, L. Suresh, and T. Xu. 2022. Automatic reliability testing for cluster management controllers. In *Proceedings of the 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI '22)*, 143–159.

[65] S. D. Palma, D. Di Nucci, F. Palomba, and D. A. Tamburri. 2020. Toward a catalog of software quality metrics for infrastructure code. *Journal of Systems and Software* 170 (2020), 110726.

[66] S. D. Palma, D. Di Nucci, and D. A. Tamburri. 2020. AnsibleMetrics: A python library for measuring infrastructure-as-code blueprints in ansible. *SoftwareX* 12 (2020), 100633. 2352–7110

[67] M. Wurster, U. Breitenbücher, L. Harzenetter, F. Leymann, and J. Soldani. 2020. TOSCA lightning: An integrated toolchain for transforming TOSCA light into production-ready deployment technologies. In *Advanced Information Systems Engineering*. N. Herbaut and M. La Rosa (Eds.), Springer International Publishing, Cham, 138–146.

[68] Indika Kumara, Zoe Vasileiou, Georgios Meditskos, Damian A. Tamburri, Willem-Jan Van Den Heuvel, Anastasios Karakostas, Stefanos Vrochidis, and Ioannis Kompatsiaris. 2020. Towards semantic detection of smells in cloud infrastructure code. In *Proceedings of the 10th International Conference on Web Intelligence, Mining and Semantics (WIMS '20)*. ACM, New York, NY, 63–67.

[69] Thodoris Sotiropoulos, Dimitris Mitropoulos, and Diomidis Spinellis. 2020. Practical fault detection in puppet programs. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering (ICSE '20)*. ACM, New York, NY, 26–37.

[70] Eduard van der Bent, Jurriaan Hage, Joost Visser, and Georgios Gousios. 2018. How good is your puppet? An empirically defined and validated quality model for puppet. In *Proceedings of the IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 164–174. DOI: https://doi.org/10.1109/SANER.2018.8330206

[71] Marcus Pendleton, Richard Garcia-Lebron, Jin-Hee Cho, and Shouhuai Xu. 2016. A survey on systems security metrics. *ACM Computing Surveys* 49, 4 (Dec. 2016), Article 62, 35 pages. DOI: https://doi.org/10.1145/3005714

[72] Werner Heijstek, Thomas Kühne, and Michel R. V. Chaudron. 2011. Experimental analysis of textual and graphical representations for software architecture design. *Proceedings of the International Symposium on Empirical Software Engineering and Measurement*, 167–176.

[73] Luca Allodi, Marco Cremonini, Fabio Massacci, and Woohyun Shim. 2020. Measuring the accuracy of software vulnerability assessments: Experiments with students and professionals. *Empirical Software Engineering* 25 (2020), 1063–1094. DOI: https://doi.org/10.1007/s10664-019-09797-4

[74] Luca Allodi, Silvio Biagioni, Bruno Crispo, Katsiaryna Labunets, Fabio Massacci, and Wagner Santos. 2017. Estimating the assessment difficulty of CVSS environmental metrics: An experiment. In *Future Data and Security Engineering*. Tran Khanh Dang, Roland Wagner, Josef Küng, Nam Thoai, Makoto Takizawa, and Erich J. Neuhold (Eds.), Springer International Publishing, Cham, 23–39.

[75] Katsiaryna Labunets, Federica Paci, Fabio Massacci, and Raminder Ruprai. 2014. An experiment on comparing textual vs. visual industrial methods for security risk assessment. *Proceedings of the IEEE 4th International Workshop on Empirical Requirements Engineering (EmpiRE '14)*. DOI: https://doi.org/10.1109/EmpiRE.2014.6890113

Received 19 December 2023; revised 3 July 2024; accepted 9 July 2024