Journal of Software Engineering Research and Development, 2025, 13:6, doi: 10.5753/jserd.2025.4378

Strategies to Mitigate Configuration Differences in Software Development: A Rapid Review of Grey Literature

Marcos Nazário **Federal University of Pará** ***marcosnazario@iec.gov.br***
Rodrigo Bonifácio **University of Brasília** ***rbonifacio@unb.br***
Cleidson R. B. de Souza **Federal University of Pará** ***cleidson.desouza@acm.org***
Fernando Kenji Kamei **Federal Institute of Alagoas** ***fernando.kenji@ifal.edu.br***
Gustavo Pinto **Federal University of Pará** ***gpinto@ufpa.br***

Abstract Context: Configuration differences between development and production environments can lead to system failures, data loss, and security vulnerabilities. Practitioners have acknowledged the need to handle these differences. This study aims to provide insights and best practices for mitigating configuration differences between development and production environments in software development through a rapid review (RR) of Grey Literature (GL). **Methods:** The research method for data collection involves a Rapid Review of the GL. It also employs a thematic analysis to understand the collected data. **Results:** The study identified nine strategies that help practitioners to mitigate configuration differences between development and production environments, including credentials of access. Examples of these strategies include Automated Deployment Pipeline, Configuration Management Plan (CMP), and PaaS Deployment. We also provide insights into how these strategies contribute to reducing the risk of configuration-related issues and ensuring the smooth and reliable operation of software systems. Additionally, it suggests potential cost-saving methods for mitigating configuration differences. **Conclusions:** This study emphasizes the importance of understanding the strategies employed to mitigate the risks associated with managing configuration differences in software development projects. By delivering *practical* guidance and insights, this study has the potential to help IT operations and software development projects to have a deeper comprehension of software configuration management strategies and best practices.

Keywords: *Software Configuration management, Software development, Rapid Review, Grey literature, Mitigation strategies*

1 Introduction

In the dynamic landscape of software development, the inherent divergence between specialized teams working during an application's lifecycle creates a notable challenge: the development team's pursuit of agility clashes with the operation team's emphasis on stability, leading to increased time to market and a detrimental impact on software quality and product value (Mazyar, 2018). This conflict is rooted in the essential tension between developers striving for rapid changes and information technology (IT) operators tasked with maintaining infrastructure configuration (Leite et al., 2019). Notably, operational teams often exhibit reluctance to adopt new changes, causing a disparity in the adoption of new code between developers and operators (Leite et al., 2019).

To address this fundamental challenge, the concept of DevOps arises as a transformative force. Derived from the fusion of "Developers" and "Operations", DevOps represents not only a methodology but also a culture and set of practices designed to mitigate the separation between development and operations teams. It aims to enhance software delivery speed and reliability by fostering collaboration and unifying tasks through cross-functional teams (Leite et al., 2019). Technologically, DevOps leverages code-based tools and continuous delivery to achieve a high degree of automation, resulting in a smoother workflow, improved service quality (Ebert et al., 2016), and enhanced change management performance.

Within the realm of DevOps, Infrastructure as Code (IaC) stands out as a pivotal technique. IaC automates software deployment by controlling IT infrastructure through code files and replacing manual configurations with interactive tools. IaC programs, treated as source code, enable versioning, debugging, updating, and reviewing, aligning with established software development practices (Guerriero et al., 2019). In the context of substantial efforts to modernize applications through agile practices, the DevOps culture, and containerization, the complexity of software environments has escalated significantly. Teams encounter new challenges as they transition applications to production, especially in distributed settings across multiple infrastructures, clusters, and clouds (Monclus, 2021). Achieving consistency between development and production environments becomes intricate due to the complexity of control configurations.

Despite the rise of DevOps culture to bridge this gap between development and production environments and the adoption of techniques like IaC, we found that the scientific literature lacked a comprehensive exploration of the specific strategies used to align these environments. For this reason and due to the fact that IaC is a practical topic of software engineering, the use of Grey Literature (GL) as a source of evidence seems to be more beneficial to our investigation. For instance, Kamei (Kamei et al., 2020) declared that *"...there are several motivations for Brazilian SE researchers use GL, mainly because its content provides important information to researchers. However, it is still hard to find reliable information for scientific research".*

This study is the first that aims to delve into the existing gap between development and production environments, seeking alternative approaches to bridge this divide and enhance the efficiency of software delivery and operations through a Rapid Review (RR) of Grey Literature (GL). As our research topic is dynamic, a full Systematic Literature Review (SLR) could take a long time to be concluded, and its results could be out of date to be published, which influenced our decision to conduct an RR (Cartaxo et al., 2018), i.e., to gather and present the most recent practices and problems within the industry with the assistance of a quick review. Furthermore, we realized that a large portion of the most recent research in this field is found in GL, which includes technical reports, and blog posts that may take months or even years to appear in peer-reviewed journals. We may access this mine of up-to-date useful information that practitioners are actively using and discussing by concentrating on a rapid study of GL (Garousi and Felderer, 2017). We recognize that this approach could lead to drawbacks, which we discuss in detail in Section 10. Our main findings are the following:

- Nine strategies were identified that directly address and mitigate configuration differences between development and production environments. Each strategy plays a unique role in this mitigation, for instance:
  - Automated Deployment Pipelines ensure consistent, automated deployments across environments, reducing the risk of human error during manual deployments;
  - Configuration Management Plans (CMPs) provide a structured approach to managing configuration versions and changes, ensuring that both environments maintain synchronized configurations;
  - PaaS Deployments offer cloud-based platforms that standardize infrastructure, reducing environmental differences by creating uniform deployment environments;
  - Secrets Management Platforms centralize and secure sensitive information (like credentials), ensuring consistent access and preventing configuration drift between environments; and
  - Other strategies, such as Database Table Configuration, Operating System Configuration, and Web Server Configuration, focus on maintaining consistent settings and dependencies across environments, ensuring that the same infrastructure and data setups are mirrored in both environments.
- This study offers a thorough analysis of the ways in which these nine strategies help reduce the risk of configuration-related problems, ensuring the seamless and reliable operation of software systems in a variety of contexts; and
- Finally, the study also highlights the useful applications of these strategies in real-world situations while providing insights into the practices, processes, and challenges experienced by IT specialists and stakeholders involved in the management and deployment of software systems.

This work is structured into thirteen sections, designed to guide readers through a comprehensive exploration of the study. It begins with Section 1 (Introduction), setting the context, followed by Section 2 (Theoretical Background), which provides conceptual knowledge. Section 3 (Research Questions) frames the study's objectives, while Section 4 (Research Method) details the data collection and analysis processes. Section 5–8 presents the findings for each research question. Section 9 (Discussion) explores implications, while Section 10 (Limitations) acknowledges constraints. Section 11 (Related Works) contextualizes the study within the existing literature. Section 12 (Conclusions) summarizes key findings and contributions, and finally, Section 13 (Data Availability) ensures transparency by informing readers how to access supporting data. This structure ensures a logical flow from context and methodology through findings and implications, providing a thorough and transparent presentation of the research.

2 Theoretical Background

Software Configuration Management (SCM). This is the field dedicated to managing and controlling software system modifications. The configuration of development and production environments is included in this extensive area, which guarantees that these environments stay consistent, particularly as software evolves. SCM is specifically focused on the challenges that come with managing software, in contrast to traditional configuration management Tichy (1995), which originally emerged from hardware systems. Ensuring consistency between development and production environments' configurations is a significant difficulty in contemporary software development. The platforms, operating systems, dependencies, and runtime services used in these settings frequently diverge. When software has to work over hundreds or thousands of services or be spread across multiple platforms, it becomes more difficult; this is a common scenario in cloud-based and microservice designs. The software industry has embraced procedures that bring the development and production environments closer together in an effort to reduce this complexity, which has given rise to the DevOps field Leite et al. (2019). The primary objective of DevOps is to close the gap between *operations* teams, who favor stability, and *development* teams, who prioritize agility. DevOps encourages the adoption of consistent environments and configurations by utilizing automation tools and continuous integration/continuous delivery (CI/CD) pipelines. This reduces the risks related to inconsistencies between development and production environments.

DevOps. Ebert et al. (2016) defines DevOps as a culture change toward cooperation between operations, quality assurance, and development. It is about provisioning and developing business processes quickly and flexibly. DevOps effectively unifies operations, delivery, and development, enabling a lean, fluid coupling of these formerly isolated silos. It integrates the domains of development and operations through automated development, deployment, and infrastructure monitoring. Rather than working in isolation in silos, cross-functional teams now concentrate on continuous operational feature releases. This strategy helps to minimize problems caused by miscommunication among team members, deliver value more rapidly and consistently, and solve problems more swiftly.

CI/CD. Continuous Integration (CI) and Continuous Deployment (CD), are referred to as continuous methods that help organizations speed up the development and delivery of software components without sacrificing quality. In the context of software development organizations, CI is a well-established development practice (Fitzgerald and Stol, 2017). It involves team members integrating and merging development work (e.g., code) regularly, perhaps several times a day. Taking it a step further, the CD practice continuously and automatically distributes the program to client or production environments. The definition and distinction between continuous deployment and continuous delivery remain controversial in academic and industry circles (Fitzgerald and Stol, 2017). A production environment (i.e., actual customers) distinguishes continuous deployment from continuous delivery: the objective of continuous deployment practice is to automatically and gradually deploy every change into the production environment. These practices help organizations to release new features and products regularly.

Grey Literature. This concept has many definitions. Nevertheless, the most known is called Luxembourg's definition that states *"Grey Literature is produced on all levels of government, academics, business, and industry in print and electronic formats, but which is not controlled by commercial publishers, i.e., where publishing is not the primary activity of the producing body."* Garousi et al. (2019). Focusing on Software Engineering Research, recently, Garousi et al.proposed the following definition: *"Grey literature can be defined as any material about SE that is not formally peer-reviewed nor formally published."*. These definitions indication a wide concept of what would be considered a GL, indicating that it can be produced in different ways. Nevertheless, it may lead to a misunderstanding. For this reason, Adams et al. presented some terms to distinguish the different concepts about grey, including grey literature, grey data, and grey information. The term "grey data" describes user-generated web content (e.g., tweets, blogs, videos). The term "grey information" is informally published or not published (e.g., meeting notes, emails, personal memories). However, SE literature hardly distinguishes these terms. Similarly, we considered all forms of grey data and grey information as GL in our work. Further, the GL types Adams et al. were categorized GL according to "shades of grey". SE, Garousi et al. adopted these shades according to three tiers, as shown in Figure 1. In this figure, on the top of the pyramid is the "traditional literature" with scientific articles from conferences and journals. On the rest of the pyramid are what we call three tiers of GL. These tiers are run according to two dimensions: **Control** and **Expertise**. The first dimension runs between extremes "low" and "higher" and the second runs between extremes "unknown" and "known". The darker the color, the less moderated or edited the source in conformance with explicit and transparent knowledge creation criteria. We will present the Rapid Review methodology in section 4, after we discuss our research questions.

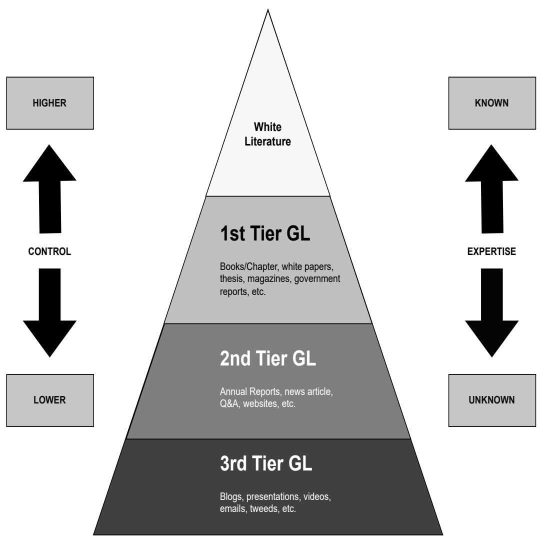
HIGHER
CONTROL
LOWER
White Literature
1st Tier GL
Books/Chapter, white papers, thesis, magazines, government reports, etc.
2nd Tier GL
Annual Reports, news article, Q&A, websites, etc.
3rd Tier GL
Blogs, presentations, videos, emails, tweeds, etc.
KNOWN
EXPERTISE
UNKNOWN

Figure 1. The "shades" of grey literature, adapted from Garousi Garousi et al. (2019)

3 Research Questions

This research aims to find out about the strategies that software developers used to mitigate the configuration differences between development and production environments. Specifically, we focus on investigating the following research questions:

RQ1: What are the strategies to mitigate differences between the development and production environments?
RQ2: Are these strategies sufficient to mitigate differences between the development and production environments?
RQ3: What are the perceived benefits of automation in development and production environments?
RQ4: What are the perceived difficulties in the automation process in development and production environments?

Answering **RQ1** is important for understanding the strategies practitioners leverage to mitigate the differences between the development and production environments. In particular, knowing that the DevOps culture is an important tool to decrease the silos between the infrastructure and development teams (Mazyar, 2018), we sought to answer this question with professional perspectives available in the GL.

Answering **RQ2** is relevant to understanding the efficacy of the strategies in reducing the differences between the development and production environments. Our study aims to identify potential gaps in the strategies found and identify opportunities for improvement by examining their effectiveness.

Answering **RQ3** allows us to understand the potential benefits of using automation to configure development and production environments. By investigating the benefits derived from automation, the study aims to uncover the tangible outcomes and value propositions associated with implementing automated processes in software configuration.

Finally, answering **RQ4** is appropriate for researchers and practitioners to comprehend the potential open challenges for a seamless replication of development and production environments. The expected outcomes of this question aim to include a deeper understanding of the barriers to successful automation, insights into common challenges encountered by software teams, and recommendations for overcoming these difficulties to streamline the automation process effectively.

4 Research Method

To address our research questions, we conducted a Rapid Review (RR) (Cartaxo et al., 2018) of Grey Literature (GL) (Garousi et al., 2019) for data collection, in conjunction with a Thematic Analysis (Cruzes and Dyba, 2011) for data analysis. We refer to our set of references as R1–R110. Figure 2 provides an overview of our research method, which involves five main steps: search process, source selection, study quality assessment, data extraction, and data synthesis.

We conduct a Grey Literature Review (GLR) for this study due to three important reasons. Firstly, as mentioned by Kamei et al. (2020, 2021), a GLR is appropriate due to the nature of the research topic since it contains reports, working papers, and industry publications, frequently containing insights and strategies that may not be found in conventional academic literature. Given the focus on solutions for software development challenges, tapping into GL documents was essential. Secondly, we decided to initiate a GLR instead of a Systematic Literature Review (SLR) because our objective was to capture current, practical insights and strategies that are actively being used in the industry but may not yet be fully documented or validated in peer-reviewed academic literature. By using this method, we were able to gather innovative techniques and new patterns Finally, conducting interviews alone may not have provided a comprehensive overview of the diverse strategies employed in the industry to mitigate configuration differences. By leveraging a GLR, the study could collect a wide range of documented strategies and insights from various documents, improving the richness and depth of the findings.

In addition, gathering timely and relevant insights from the industry was the motivation behind the decision to initiate an RR, especially in a field that is changing quickly like software development. Traditional SLRs require an extended period, and at the time results have been published, they may not be recent. By choosing an RR, we hope to capture the most recent practices and challenges that practitioners are facing, making the insights offered both pertinent and useful. This method makes it possible to apply research results more quickly in practical contexts, which is very helpful for our target audience of software developers and DevOps professionals who need fast solutions. Because of this, the RR methodology was required to keep up with the industry's rapid changes and the pressing need for practical ways to mitigate configuration differences.

4.1 Search Process

According to Garousi et al. (2019), the definition of search terms to be used in search engines needs special consideration due to the general lack of consistency in SE terminology and the possibility that this issue could even be more serious for the GL. Therefore, we followed their recommendation to perform an informal pre-search to find different synonyms

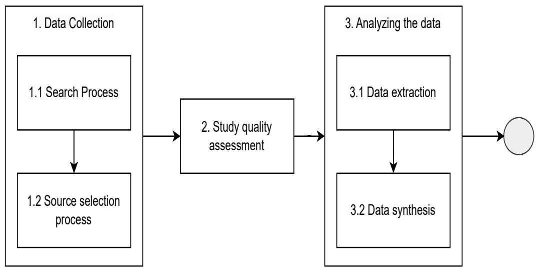

Figure 2. Research workflow.

for specific topics and consult bodies of knowledge, such as the SWEBOK for SE in general terms. We followed the protocol proposed by Cartaxo et al. (2018) to perform the RR, i.e., we must address two main concerns here: where to search and when to stop the search procedures. To gain insights into our specific search process and query strings, refer to the sheet named "String Query" and "Systematic Mapping" in our spreadsheet dataset (Nazário et al., 2024).

4.1.1 Where to search

Aiming to carefully identify platforms, databases, and archives that are most likely to yield valuable GL, we considered using all the approaches suggested by Garousi et al. (2019) as input in our research. However, in the end, we did not use the "*Contacting individuals directly or via social media*" approach since we did not see the need for that.

General web search engines play a crucial role in the research process, involving using relevant terms and phrases associated with the research topic. Platforms like Google provide a gateway to the vast information available online. However, employing effective search strategies becomes essential to filter and select high-quality, pertinent documents from this extensive data pool.

Another avenue for exploration is the examination of reference lists within existing documents, websites, and articles. This approach adds depth to the research by uncovering interconnected documents and establishing a comprehensive understanding of the subject.

In addition to general searches, specialized databases, and websites offer a more structured and organized approach to accessing reports and documents related to software companies' strategies for configuration mitigation.

4.1.2 When to stop the search

Our search process involved considering the top 100 search engine hits as a pragmatic stopping condition based on the likelihood that these top results would contain high-quality, authoritative, and well-cited documents. Figure 3, summarizes the RR process executed in this study, which we describe next. In the first round, the first author analyzed the document title and keywords and excluded those that did not meet the criterion defined. In the second round, the first author analyzed the document resumes when available. In the third round, the first author analyzed the entire document content and excluded those that could not answer any research question. In the fourth round, the first author performed a snowballing process when it was available to avoid possible missing documents as recommended by Wohlin (2014).

The last author supervised all these rounds to avoid bias in the selection. This approach was made to guarantee that the selection process remained impartial and free of any potential biases that might have affected the initial author's judgments. By adding another level of examination and objectivity, having the final last author monitoring the entire process was intended to improve the selection process's reliability and integrity.

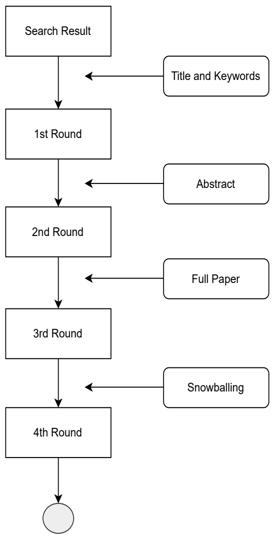

Figure 3. RR Procedure.

4.1.3 Document Selection Process

The source selection criteria are intended to identify documents that provide direct evidence about our research questions. It comprehends the definition of inclusions and exclusions criteria. These criteria were also applied when performing the forward and backward snowballing sampling method (Wohlin, 2014). This resulted in the following inclusion and exclusion criteria:

We include documents that openly discuss software companies' or professionals' strategies to mitigate configuration differences between development and production environments. We include documents that are relevant to the study, such as materials published within the last ten years, as they are more likely to reflect recent trends and best practices. Our documents are written in English, Portuguese, or Spanish.

We exclude documents that do not provide specific information on the strategies used to mitigate configuration differences between development and production environments. We also exclude significantly outdated documents, as they may not reflect current industry practices and trends. Finally, we excluded documents that would require some payment for access.

4.1.4 Study quality assessment

According to Garousi et al. (2019), the quality assessment of documents is about defining the scope to which a document is valid and free of bias. Unlike traditional peer-reviewed literature, GL tends to be more eclectic and less controlled. Therefore, the quality of GL is frequently more laborious to assess. A variety of models exist for the quality assessment of GL documents. The adoption of Garousi's study quality assessment checklist (Garousi et al., 2019) in this research study was driven by its established framework, comprehensive metrics, relevance to GL, promotion of consistency and comparability, validation and reproducibility, the requirement of guidance for researchers, and enhancement of methodological robustness. We marked each quality assessment metric as "1" if the document satisfied it and "0" otherwise. Next, we present the metrics we used to assess the quality of the references.

- **Authority of the producer.** It assesses the credibility of the entity or individual responsible for creating the GL. High authority suggests that the document is reputable, reliable, and likely to provide trustworthy information. For example, a report produced by a well-known research institution may be considered authoritative.
- **Methodology.** It examines the research methods, processes, and data collection techniques employed in producing the GL. A robust and well-documented methodology improves the document's reliability and is valuable for understanding how the information was obtained.
- **Objectivity.** It assesses whether the GL documents carry a neutral and unbiased perspective. Objective documents deliver information without significant bias or opinion, which is critical for research validity.
- **Date.** It evaluates the recentness of the GL. Typically, more recent documents are preferred, especially in fast-evolving fields, as they likely reflect current trends, practices, and knowledge.
- **Position w.r.t. Related Sources.** It concerns assessing how the GL documents relate to existing or related documents. Understanding its position within the literature landscape is essential for evaluating its contribution.
- **Novelty.** It assesses whether the GL documents present new ideas, concepts, or findings not widely documented in existing literature. Novel documents can be beneficial for advancing research.
- **Outlet Type.** The kind of outlet or publication medium for the GL document is considered. Counting on the research context, certain outlets, such as government reports, industry publications, or academic working papers, may be selected for their reliability and relevance. It also is organized into three tiers.
- **Impact.** It analyzes the influence or significance of the GL document in the field. High-impact documents are indicative of valuable contributions.[1]

After collecting the metrics, we must aggregate them into a single metric. This requires a normalization process to ensure that each metric contributes meaningfully to the overall assessment. We used Z-score normalization (Patro and Sahu, 2015) for such aggregation, transforming each metric into a standard score and then combining them into a single composite metric.

Once the impact metrics have been normalized, they can be aggregated by simply taking the normalized values' av-

[1]For counts of social media shares, we used `http://www.sharedcount.com`. For the other counters, we used `http://www.seoreviewtools.com/valuable-backlinks-checker/`

erage. A higher Z-score indicates that the GL has an above-average impact compared to the average and standard deviation across these metrics. Finally, we provide notes for each row for justifying each criterion. The sum of the criteria and the last normalized values (between 0–1) show the quality assessment outcome for each GL document, out of a total quality score of 20 (the total number of individual criteria). Ultimately, all normalized document values averaged 0.46, representing a mid-average impact. The sheet named "GL Quality" in our dataset (Nazário et al., 2024) presents this data. Our data collection procedures resulted in 110 GL documents.

4.2 Analyzing the data

In this section, we detail our data analysis procedures, which involve steps from data extraction to data synthesis. To gain insights into our specific data analysis procedures, including examples, refer to our spreadsheet dataset (Nazário et al., 2024).

4.2.1 Data extraction

The documents in our pool were examined with no particular RQ in mind, aiming only to collect both quantitative and qualitative data. However, we were unable to answer each RQ based on the incomplete extracted data, necessitating further work to examine, read, and extract the missing data from the codes and themes once more.

According to Ogawa and Malen (1991), since documents in the GL are frequently written for non-academic purposes and audiences and because documents often address different aspects of a phenomenon with diverse degrees of diligence, it is paramount that researchers record the purpose and specify the coverage of each GL document. These items were also in our quality assessment checklist in Section 4.1.4. Still, Garousi and Felderer (2017) experienced that GL documents have a less standardized structure than formal literature. Therefore, it is helpful to include "traceability" links (i.e., comments) in the data extraction form to point to the exact location in the GL document where the extracted information was located. We have created spreadsheets (hosted on Google Docs) with direct traceability to GL's research questions in mind, based on Garousi and Felderer (2017) recommendations to make the design of data extraction forms easier. It is available in the sheet named "Systematic Mapping" in our spreadsheet dataset (Nazário et al., 2024).

The **Row ID** column provides a unique identifier for each entry in the table. This identifier is used to reference the entry within the table and to link it to other documents. It can appear repeatedly to represent other column values associated with the same code entry. The **Source of Search (SS)** column refers to a unique row identifier from Table 1, where a particular GL document was retrieved or identified. It can refer to only one row identifier. The **RQ1** column refers to a unique row identifier associated to the RQ1. It can refer to only one row identifier. The **RQ2** column refers to a unique row identifier from Table 5 associated with RQ2. It can refer to only one row identifier. The **Year** column indicates the year of publication of the document. This information can be used to assess the currency of the information in the entry. Sometimes, the year of publication is not visible. For these entries, we used the following sites to discover them: https://archive.org/web/ and https://carbondate.cs.odu.edu/. The **Tier** column represents the level of importance of the entry. The tiers range from 1 to 3. The **Note** column provides additional information about the entry, such as specific quotes or relevant keywords. This information can help to further clarify the relevance of the entry to the research questions.

4.2.2 Data synthesis

We employed a mixed-method approach based on qualitative and quantitative data analysis methods. We follow a qualitative approach through Thematic Analysis Cruzes and Dyba (2011) to analyze our data. This procedure could be summarized as follows: Extract data, Code data, Translate codes into themes, Create a model of higher-order themes, and Assess the trustworthiness of the synthesis. To complement this qualitative analysis, we used descriptive statistics to discuss the frequency and distribution of the data.

4.2.3 Extract and Code data

During this process, we created one global artifact, **the comparison sheet**, to keep all the **codes**. The first author analyzed the documents from GL while discussing pertinent codes with the last author. All the initial codes, open coding, axial coding, and selective coding are available in the sheet named "Thematic Analysis" in our spreadsheet dataset (Nazário et al., 2024). Also, see Figure 4.

1. *Data Collection*. We extracted relevant text segments or passages from the GL pertinent to our research questions.
2. *Open Coding*. It involves breaking down the data into smaller units and assigning initial codes based on their content. We conducted open coding by thoroughly reading the documents and identifying 40 initial codes.
3. *Axial Coding*. Concerns identifying relationships between categories and subcategories. We performed axial coding to explore relationships and connections between the initial codes. We identified 19 categories of codes.
4. *Selective Coding*. Includes selecting the most significant categories and subcategories to develop a coherent narrative or theoretical framework. We decided to keep only the codes with more than ten percent relevance and codes that focus on core categories that represent the main focus of the study, resulting in 15 refined categories of codes. We filled the cells with concise statements, with columns representing participants and rows representing the summarized codes as we show in Table 2.

Table 1. Types of Source of Search.

Row ID	Source of Search (SS)	Meaning
1	Contacting individuals directly	DM/Email/Tweet
2	General web search engines	Google/Blog
3	Reference lists and backlinks	Snowball method
4	Specialized databases and websites	StackOverflow/Forum

5. *Analysis and Interpretation*. We analyzed the coded data to draw conclusions and insights into the benefits and difficulties associated with different strategies for managing configuration differences between development and production environments in software development.

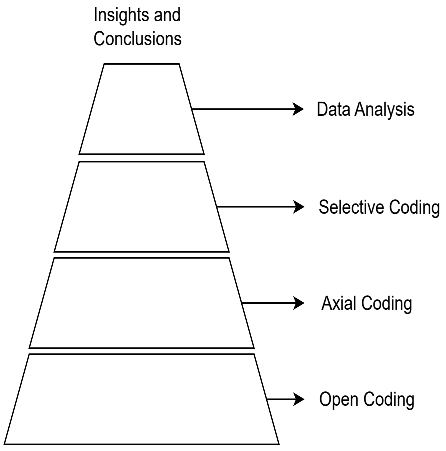

Figure 4. Extract and Code data.

4.2.4 Translate codes into themes and Create a model of higher-order themes

Translating codes into themes essentially involves understanding how the various codes from Table 2 might come together to generate a larger theme. The authors searched for patterns, relationships, and similarities among the codes before classifying them into themes. These theme names were chosen with the intention that they should convey the overarching concept that "connects" the set of codes contained inside them. The procedure came to an end when the authors covered all the potential themes that had emerged from the data. The generated themes are listed and discussed below:

The theme **Access Control and Security** deals with controlling access privileges in various contexts. It concerns data protection and security. Access control is essential for preserving the accuracy and security of data and making sure that only people with the proper authorization may access the environments. This theme is associated with the code: Access restriction between environments.

The theme **Data and Environment Management** focuses on whether data and settings are consistent or inconsistent across various environments inside a system. It addresses the necessity for preserving data integrity and making sure that settings are consistent across environments. This theme is associated with the following codes: Consistency of data and settings between environments, Dependencies between environments along the pipeline, Inconsistency of data and settings between environments, and Lack of visibility into changes.

The theme **Infrastructure Provisioning** concentrates on the techniques and tools used to set up and manage the infrastructure required for software development and deployment are referred to as infrastructure provisioning. Infrastructure

Table 2. GL selective codes sorted by their descending average of occurrences.

Codes	AVG (%)
Time-saving	85,00
Automated Infrastructure Provisioning	66,06
Dependencies between environments along the pipeline	61,82
Inconsistency of data and settings between environments	60,45
Access restriction between environments	55,76
Consistency of data and settings between environments	48,18
Monitoring Mechanisms	36,82
Infrastructure Manual Provisioning	28,64
Control of financial expenses	25,91
Lack of visibility into changes	25,91
Technical debt	25,91
Reduced manual effort and errors	18,64
Build Validation Mechanisms	16,36
Team communication mechanisms	15,91
Well-defined scope of action between teams	14,09

provisioning for the development, testing, and deployment environments is included, both manually and automatically. Because it affects the consistency and dependability of the environments, infrastructure provisioning is directly related to data and environment management, as well as validation and quality assurance. This theme is associated with the code: Automated Infrastructure Provisioning, Infrastructure Manual Provisioning, and Reduced manual effort and errors.

The theme **Monitoring and Performance** emphasizes that monitoring systems are necessary for keeping track of and evaluating the functionality and state of the system or application. Since monitoring offers insights into data and environment management, validation, teamwork, infrastructure provisioning, and access control, it connects to every other subject. It supports problem identification, performance optimization, and system integrity, and is associated with code Monitoring Mechanisms.

The theme **Team Collaboration and Communication** focuses on effective team collaboration and communication. Each team's scope of work must be well-defined, collaboration must run smoothly, and teams must not be kept apart from one another. To successfully solve issues with data and environment management, validation, and infrastructure provisioning, collaboration and communication are essential. Related codes are Team communication mechanisms, Time-saving, and Well-defined scope of action between teams.

The theme **Validation and Quality Assurance** encompasses validation procedures and technical debt management that typically refers to compromises made during the software development process that may impact the overall quality, maintainability, and efficiency of the codebase. As it involves procedures and methods to guarantee the quality, correctness, and dependability of the system or product being built, it is strongly tied to data and environment management. Validation aids in locating and resolving problems with data and environment consistency. This theme is related to code: Build Validation Mechanisms, Control of Financial Expenses, and Technical Debt. All these themes are available in the sheet named "Thematic Analysis" in our spreadsheet dataset (Nazário et al., 2024).

These themes that surfaced could be investigated and analyzed further to create a model consisting of higher-order themes and the associations among them.

Presenting the study's findings coherently and logically requires the classification of themes and second-order themes. We intended to provide a more structured framework that permits a better understanding of the complex relationships between the different elements affecting SCM in software development by establishing these categories. The second-order themes explore specific elements of the primary themes, allowing a more comprehensive examination, while the primary themes indicate broad areas of concern. In addition to helping determine patterns and correlations in the data, this hierarchical approach enhances the clarity of the findings, which makes it simpler for researchers and practitioners to apply what they have learned into practice. In the end, this classification helps to show how many methods and practices are related to one another, highlighting the necessity of a comprehensive approach to successfully manage configuration differences. As such, we grouped the previous themes into two new ones: **Data Governance and Management** and **Operational Efficiency and Effectiveness**.

The second-order theme **Data Governance and Management** focuses on organizations that assure the availability, correctness, integrity, and security of data by putting effective data governance and management procedures into place. This encourages productive and efficient operations by supplying trustworthy data for decision-making, process optimization, and performance monitoring. This is associated with the following categories: Access Control and Security, Data and Environment Management, Validation and Quality Assurance.

The second-order theme **Operational Efficiency and Effectiveness** concentrates on access to high-quality data that is essential for organizations to improve operational success and efficiency. Consistent, accurate, and secure data management is ensured by effective data governance and management. This makes it possible to reduce procedures, improve workflows, and make wise decisions, which results in operational excellence. This is linked to the following themes: Infrastructure Provisioning, Monitoring and Performance, Team Collaboration and Communication.

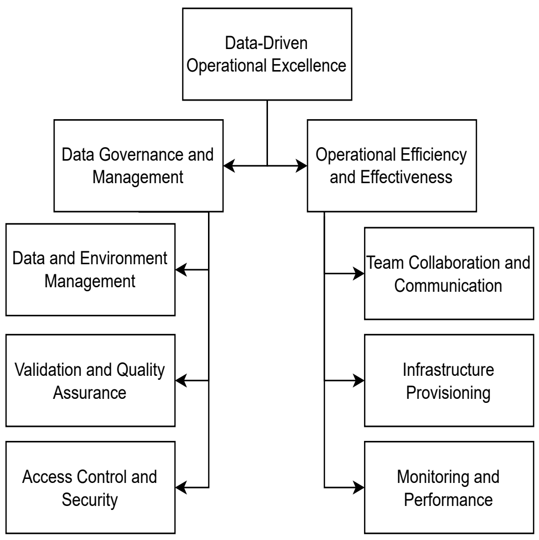
Data-Driven Operational Excellence
Data Governance and Management
Operational Efficiency and Effectiveness
Data and Environment Management
Validation and Quality Assurance
Access Control and Security
Team Collaboration and Communication
Infrastructure Provisioning
Monitoring and Performance

Figure 5. Model of higher-order themes.

Indeed, these second-order themes are interdependent and symbiotically related: Operational Efficiency and Effectiveness reinforce the need for efficient data governance processes, while Data Governance and Management lay the groundwork for these operational outcomes. Consequently, we decided to group them into a single theme named Data-Driven Operational Excellence. The final structure can be seen in Figure 5. The theme of **Data-Driven Operational Excellence** emphasizes how crucial it is to use efficient data governance and management procedures that improve an organization's operational effectiveness. All these higher-order themes are available in the sheet named "Thematic Analysis" in our spreadsheet dataset (Nazário et al., 2024).

4.2.5 Assess the trustworthiness of the synthesis

Ensuring the results are trustworthy and accurately reflect the perspectives of the reviewed documents is essential. Although formal peer review is typically absent from grey literature, we emphasize that our research makes an effort to receive an extensive review to ensure its trustworthiness Guba (1981). We used a number of important methods to confirm the themes' accuracy and make sure they matched the perspectives provided in the documents. Initially, the data were carefully and systematically analyzed Cruzes and Dyba (2011) to identify themes, patterns, and connections that accurately represented the conclusions obtained from the grey literature. Both the first and last authors independently reviewed the data during the coding and theme development phases, permitting a comparison of interpretations and the identification of any potential biases. By collaborating, it was possible to make sure that the themes were deeply based on the data rather than being shaped by individual opinions. The themes and their consistency with the source material were also discussed during the peer debriefing process, in which we participated. This phase performed an essential part in validating the interpretations' accuracy and enhancing the findings' credibility. Through the use of these rigorous validation methods, we aim to increase the credibility of our findings and offer reliable insights into the approaches for configuration difference mitigation.

4.2.6 Results of the GLR

In this section, we summarize the key findings from our GLR, including the number of documents selected, the strategies identified for mitigating configuration differences, and the frequency of citations for each strategy. A total of 110 documents were selected for analysis in this GLR. These documents were obtained from various grey literature outlets, including blogs, technical reports, and industry publications, ensuring a comprehensive representation of current practices in software development. Through our analysis, we identified nine distinct strategies employed by software development teams to mitigate configuration differences between development and production environments. Table 3, presents the strategies that were found, and the number of documents that cite each strategy. We detail all strategies in the following sections, which give answers to our research questions.

Table 3. Frequency of citations of the strategies.

Strategies	Cited
Automated Deployment Pipeline	13
Build Profiles Management	10
Configuration Management Plans	12
Database Table Configuration	11
Manual Configuration	12
Operating System Configuration	13
PaaS Deployment	16
Web Server Configuration	6
Secrets Management Platform	17
TOTAL	110

5 RQ1 – What are the strategies taken to mitigate differences between the development and production environments?

Table 4 summarizes the strategies that emerged during our research. Our initial themes do not sufficiently represent this research question to provide comprehensive insights, it was necessary to conduct a new round of derived themes specific to answer this question from the theme tree. It involved revisiting the second-order themes with a fresh perspective that could represent our developed concept of **strategy**. The strategy was defined as a planned approach aimed at reducing differences between development and production environments. It typically involves a combination of practices, tools, and configurations that ensure better alignment, and fewer issues when moving software from development to production. These new **strategy themes** were independently created by the first author and reviewed by the last author to be contrasted and discussed. It was not necessary to do any kind of additional iteration or adjustments.

1. Theme - Data Governance and Management
   (a) Automated Deployment Pipeline.
   (b) Build Profiles Management.
   (c) Configuration Management Plans.
   (d) Database Table Configuration.
   (e) Manual Configuration.
2. Theme - Operational Efficiency and Effectiveness
   (a) Secrets Management Platform.
   (b) Operating System Configuration.
   (c) PaaS Deployment.
   (d) Web Server Configuration.

Table 4. Average of adoption for the Types of Strategies with their frequency

Types of Strategy	Frequency	AVG (%)
Automated Deployment Pipeline	13	11,82
Build Profiles Management	10	9,09
Configuration Management Plans	12	10,91
Database Table Configuration	11	10,00
Manual Configuration	12	10,91
Operating System Configuration	13	11,82
PaaS Deployment	16	14,55
Secrets Management Platform	17	15,45
Web Server Configuration	6	5,45

The **Automated Deployment Pipeline** strategy involves a series of automated stages. It is often integrated with a continuous integration and continuous delivery (CI/CD) system. It includes the building phase that takes the code to be compiled or interpreted and transformed into deployable artifacts. The testing phase executes tests to ensure software functionality and quality. The deployment phase delivers the software to be automatically deployed to the target environment. It has several advantages. The pipelines enable the automatic testing of configurations, reducing the risk of human error and ensuring that configurations are properly tested before deployment. These pipelines make it possible to automate the deployment process, allowing teams to quickly and easily deploy updates to environments. It allows teams to collaborate more effectively, as they can easily view and review changes to configurations. For example, the document R1 declared: *"Teams I work with use pipelines for infrastructure for the same reasons that development teams use pipelines for their application code. It guarantees that every change has been applied to each environment and that automated tests have passed. We know that all the environments are defined and created consistently."*. Nevertheless, this strategy has a few disadvantages. The pipelines can be complex and difficult to manage, especially for teams with limited technical expertise. Integrating it with existing systems can be challenging, requiring time and resources to ensure that everything is working as expected. The pipelines require continuous maintenance, as new software versions are released and bugs are discovered. For example, the document R6 declared: *"However, unnecessary job run consumes a lot of resources like time and money. The resources, tools, and infrastructure needs can often add up to the project costs. This can slow down the processes/hamper development if developers and executives are not guided/trained properly. Therefore, it is significant to maintain a constant frequency between executing continuous integration job and utilizing resources."*.

The **Build Profiles Management** strategy includes configurations or sets of instructions that specify how the software should be built for different environments or scenarios. This strategy involves defining and maintaining these profiles, ensuring consistency and efficiency across various environments. It has several advantages. It ensures that configurations are consistent across all environments, reducing the risk of inconsistencies and errors. It automates the process of configuring environments, which makes it easier to repeat the process, and reduces the chances of manual errors. It makes the process of updating configurations faster and more efficient. For example, Apache Maven structures this strategy into profiles that contain all the application configurations per environment. Once specified the profile, it selects and merges the configurations at build time, guaranteeing that all configurations will be there. For example, the document R15 stated: *"The profiles help to execute specific elements of the build when we need them. This optimizes our build process and helps to give faster feedback to developers."*. Nevertheless, this strategy has a few disadvantages. It requires specialized knowledge and technical expertise. It can need regular maintenance and support, which might put a heavy burden on IT workers. It imposes a complete rebuild of the application to apply changes in configurations. For example, the document R22 declared: *"Maven needs the maven installation in the system for working and maven plugin for the IDE. If the maven code for an existing dependency is* *not available, then one cannot add that dependency using maven.".*

The **Configuration Management Plan (CMP)** strategy involves the development and implementation of comprehensive plans and processes to manage and control software configurations throughout the development and deployment life-cycle. It encompasses documenting and tracking configuration items, establishing version control, defining change management procedures, and ensuring adherence to configuration standards. It has several advantages. It confirms that all configurations are consistent, ensuring that the same standards are used for different environments. It provides a way to track changes made to configurations, making it easier to find out the root cause of problems and resolve them. It is necessary to meet regulatory compliance requirements, such as ISO 27001, PCI DSS, and others. It can help teams collaborate better, as everyone knows exactly what to do and how to do it. For instance, these plans in the Ansible tool are called *Playbooks* and the elements of this plan are called *tasks*. As a result, teams could quickly assess the present state of the environment and decide what adjustments will be necessary. For example, the document R23 announced: *"Without the definitions in the CMP, your staff will probably document as they see fit. However, the goal must be a uniform standard that everyone can follow.".* Also, document R26 declared: *"...configuration data is critical when it comes to the operations phase of the DevOps cycle. It's easier to understand the impact of configuration management by first examining what would occur in its absence.".* Yet, this strategy has several disadvantages. It requires dedicated resources, such as IT staff and specialized software. Developing and implementing it can be time-consuming, which may reduce the time available for other relevant tasks. It can be rigid and inflexible, making it difficult to adapt to changing requirements or needs. Sometimes plans can be complex to understand, making it difficult for people to use them effectively. For example, the document R25 declared: *"However, a collection of independently-produced CM Plans accommodating the requirements of each site or contract rarely results in a harmonized CM strategy and plan that addresses the larger needs and risks of the enterprise as a whole, including all the future business uncertainties, while leveraging its core strengths which they sell to their customers.".*

The **Database Table Configuration** strategy involves managing and configuring database tables to ensure consistent and efficient storage of data within a software application. It focuses on designing and structuring database tables to align with the application's data requirements and business logic. It has several advantages. Since it can handle large amounts of data, making it suitable for large organizations with complex infrastructure. The configuration data can be stored in a centralized location, making it easier to manage and track changes by multiple teams, departments, and applications, which can help improve collaboration and communication. The changes to the configuration table can be tracked and audited, which can help maintain security and regulatory compliance. For example, the document R35 declared: *"Put in these terms, it's pretty obvious that database solutions start to really dominate text (content) the moment a config file leaves the context of a single computer and needs to be managed by multiple users.".* However, this strategy has several disadvantages. It can increase the complexity of managing data, especially for organizations without dedicated database administrators. If the database already stores large amounts of configuration data, it can result in performance overhead, which can impact the speed and responsiveness of the configuration table. For instance, the document R35 declared: *"When configuration becomes data — it hurts... At some point, when there are more VMs than story points in your next sprint, their management stops being configuration and becomes data... Data, which may change quickly and at any time... should instead live in its ancestral home, a database".* This supports the idea that having a lot of configuration data stored in a database can cause overhead and affect system responsiveness by showing the performance problems that occur when configuration data grows. Integrating configuration data with additional applications can be challenging, especially when different databases and systems are involved. Inconsistent data can lead to unanticipated behavior and errors. For example, the document R45 declared: *"It really depends on your application. Storing the settings in the database has several disadvantages: Relies on database connection and Overhead when reading from database.".*

The **Manual Configuration** strategy involves managing configurations manually. This represents the cases where the resources do not take advantage of any automation tool or standardization method to alter configuration files. It has several advantages. Manually managing configurations allows for greater flexibility in customizing and adjusting configurations to meet specific needs. By manually managing configurations, workers get hands-on experience with the configuration process, which increases their knowledge and abilities. When configurations are managed manually, people may spend time reviewing and validating each configuration change. However, this strategy has several disadvantages. It can be a time-consuming process, particularly in larger organizations where there are many configurations to manage. It is often inconsistent due to human error or oversight. It can be inefficient, particularly when compared to automated solutions that can manage configurations more quickly and accurately. It is not easily scalable, making it difficult to manage configurations in larger organizations or in rapidly growing businesses. For example, the document R49 declared: *"Manual case management comes with its own list of issues and limitations. Businesses that use manual and outdated methods to handle their cases and incidents struggle with consistency and overall productivity.".* In another example, the document R49 informed that: *"This is an area of concern when it comes to manual methods—the sheer complexity makes it difficult to keep track of who's working on what. With the lack of visibility, managers may not be able to gauge progress or make an estimation of when the case will most likely be resolved. And this can greatly impact the overall experience the customer has with the organization.".*

The **Operating System Configuration** strategy includes adjusting system parameters, enabling or disabling specific features, managing user access and permissions, setting up security measures, and optimizing resource allocation to meet the requirements of the software application through operating system (OS) settings. It has several advantages.

It is portable across different operating systems and environments, making it easier to maintain the same configuration values. Once the configuration can be kept consistent by using the same set of variables across several systems, it is consistent. It can be automated by scripting or programming the modification of these variables, making it easier to manage large-scale changes. For example, the document R60 affirmed: *"Using environment variables is one technique to make your app easier to configure by separating infrequently changing data from your code."*. Nonetheless, this strategy has disadvantages. It is a possible security risk because anyone with sufficient privileges can access it in plain text. Managing these variables can become complex, especially in large organizations with multiple systems and environments without an automation tool. For example, the document R61 declared: *"If you make one small mistake somewhere between tens or hundreds of lines of environment variables, the entire file might not be parsed, and your program will throw unrelated errors throughout."*. Although the application deployment landscape has been completely changed by containerization using tools like Docker, which encapsulates applications and their dependencies in a standardized environment, this strategy can be successfully incorporated into Docker containers. Through Dockerfiles, developers can define these environment variables directly within the container configuration by utilizing Docker's ability to modify the underlying operating system settings. This ensures that the application running in a container not only functions consistently in different environments but also follows the appropriate configurations for the operating system to achieve the greatest security and performance.

The **Platform as a Service (PaaS) Deployment** strategy involves leveraging a local or cloud-based platform to deploy and manage software applications. It eliminates the complexities of infrastructure management by providing preconfigured and scalable infrastructures. Developers package their applications into containers or code bundles and deploy them onto the PaaS platform, which handles the provisioning and management of the underlying infrastructure. It has several advantages. It allows organizations to scale their infrastructure as needed, making it easier to manage configurations across different environments. It is designed to be user-friendly, so even non-technical staff can manage configurations without needing in-depth technical knowledge. It often comes with built-in automation tools that streamline the process of managing configurations, reducing the risk of human error. It can be more cost-effective than other options for managing configurations, as they mitigate the need for in-house infrastructure and technical staff. For example, the document R73 declared: *"PaaS platforms allow you to spend more time productively, instead of spending hours building your server."*. As another example, document R74 declared: *"PaaS tools also allow businesses to analyze their data, access business process management (BPM) platforms, add communication features to applications and maintain databases."*. Regardless, this strategy has several disadvantages. It may not offer the level of customization that organizations need to meet their specific requirements, leading to limitations in their ability to manage configurations effectively. It may not provide the level of security required to protect sensitive configuration data, which can be a concern for organizations dealing with confidential or regulated data. It is dependent on the provider for updates, maintenance, and support, so organizations must trust their provider to maintain the quality of their service. It can be difficult to switch from once implemented, meaning organizations may become locked into a particular provider and their offerings. For example, the document R74 declared: *"There are always two sides to every story. While it's easy to make the case for PaaS, there's bound to be some challenges as well. Vendor Dependency: Very dependent upon the vendor's capabilities. Risk of Lock-In: Customers may get locked into a language, interface or program they no longer need. Compatibility: Difficulties may arise if PaaS is used in conjunction with existing development platforms. Security Risks: While PaaS providers secure the infrastructure and platform, businesses are responsible for security of the applications they build."*.

The **Secrets Management Platform** strategy is a specialized software designed to securely store, manage, and control access to sensitive information such as passwords, API keys, and encryption keys. It plays a crucial role in safeguarding sensitive data by providing centralized storage, access controls, and audit trails. It can be seamlessly integrated with developer workflows, these platforms support a secure and smooth transition from development to production, fostering a more robust and manageable software development lifecycle. For example, the document R94 affirmed: *"It is important to use secure secret's management practices, such as storing secrets in a secure, centralized location, and using encryption, access controls, and audit logs to manage access and usage."*. However, providing a secure means of handling sensitive information comes with inherent challenges like Vendor lock-in and regulatory compliance challenges must be addressed, and adapting legacy systems to new SMPs may be difficult. For example, the document R109 declared: *"Yeah, so it depends on– so from my experience there are still a lot of organizations that are doing traditional On-Premises legacy IT, right? So their passwords are still very static. They're not as dynamic. And so they still have to work that. Like you said, they're still working on very monolithic applications."*.

The **Web Server Configuration** strategy involves setting up and customizing web servers to manage the credentials of accesses, server settings, and web security measures to meet the specific requirements of the application. It has several advantages. The configuration data stored on the Web Server is easily accessible by multiple applications and services, making it a convenient solution for managing configurations. It can easily handle numerous requests, making it a scalable solution for managing configurations. Due to the configuration files being saved on a server that can be quickly updated. For example, Apache Tomcat has a feature named JNDI Datasource that helps with this kind of configuration. For example, the document R88 affirmed: *"This helps us in creating and using DataSource connection pool with just few lines of configuration."*. Nonetheless, this strategy has several disadvantages. Storing sensitive configuration data on a remote server can be a security concern, as there is a risk of data being intercepted or compromised. Managing and maintaining the Web Server and its configurations can be a time-consuming and resource-intensive task, especially for larger organizations without an automation tool. The configuration data format changes according to the HTTP Server, and it will be necessary to rewrite to a new format in case of migration to another HTTP Server. For example, the document R93 declared: *"First on our list is server configuration issues. These issues arise when your server is not set up correctly, making your website or service unavailable to visitors. Examples of these issues include incorrect DNS settings, proxy server misconfiguration, and misconfigured virtual hosts."*.

6 RQ2 – Are these strategies sufficient to mitigate differences between the development and production environments?

At first, we thought that RQ1 had provided us with a good collection of themes regarding the steps done to mitigate differences, but we immediately found that these themes failed to capture the key aspects of what RQ2 was requesting. We needed a way of illustrating the opinions expressed in the GL documents about the effectiveness of these strategies. Despite being useful for RQ1, our current themes were not intended to express all the details about sufficiency. We consequently took the decision to write fresh codes that are exclusive to RQ2. Using an inductive coding technique, we returned to our GL documents and looked for any mentions or indications of how sufficient the strategies were considered to be. This process led us to create four new codes: **Sufficient**, **Insufficient**, **Partially Sufficient**, and **Not specified** measurements. It was not necessary to do any kind of additional iteration or adjustments. We are aware that coming up with new codes could appear like a strange step, particularly considering that we had already created a number of themes for RQ1. However, we believed that this approach was needed to fully address RQ2 and offer a comprehensive answer.

The "Sufficient" category refers to documents enforcing that the current infrastructure management practices adequately address the configuration differences between development and production environments. These documents argue that their company's processes, tools, and technologies are helping them mitigate the differences and ensure their systems are running smoothly and reliably. The "Partially Sufficient" category refers to documents suggesting that the current infrastructure management practices adequately address the configuration differences. However, there are still some areas that need improvement. The "Insufficient" category refers to documents arguing that the current infrastructure management practices are not adequate in addressing the configuration differences, and they believe that their company's processes, tools, and technologies are not helping them to mitigate the differences and ensure their systems are running properly. The "Not specified" category refers to documents that could not provide a clear opinion on the current infrastructure management practices, either because they are unfamiliar with or have not seen them in practice. Table 5 summarizes the results that emerged during our GL research.

Table 5. The perception of satisfaction by employed mitigation strategies found in GL.

Strategies	Sufficient	Partially Sufficient	Insufficient	Not Specified
Automated Deployment Pipeline	8	3	-	2
Build Profiles Management	6	4	-	-
Configuration Management Plans	8	3	1	-
Database Table Configuration	6	1	4	-
Manual Configuration	6	4	2	-
Operating System Configuration	7	6	-	-
PaaS Deployment	8	8	-	-
Web Server Configuration	3	3	-	-
Secrets Management Platform	11	6	-	-
PERCENT	63%	38%	7%	2%

The "PERCENT" row summarizes the overall satisfaction with the mitigation strategies, showing that 59% were perceived as sufficient, 32% as partially sufficient, 7% as insufficient, and 2% were not specified in the documents from Grey Literature.

For those documents that considered the strategies insufficient, we realized that even if the strategy is good, the way it has been applied leaves room for areas that might be enhanced when used together with other automation tools. These areas contribute to disappointment with the current strategy in use. For example, the document R6 agrees with the adopted strategy: *"To sustain in today's IT market and to create credibility, companies will have to focus on quicker release and quality product. It is equally important to improve or update products timely through a reliable delivery process. In order to achieve all these, implementation of CI/CD becomes inevitable."*.

Furthermore, document R37 declared: *"Maybe, just maybe, there's not a single approach to managing applications and systems that is actually "correct" at all scales. Maybe, as much as I hate to admit it, both IaC and colorful click-button-do-thing tools have their place in modern tech companies. Maybe there's a breakpoint where configuration becomes data, after which point our lack of tight-gripped control over configuration matters less than easy self-service UX for users."*.

7 RQ3 – What are the perceived benefits of automation in development and production environments?

For the research questions RQ3 and RQ4, we relied on the original thematic analysis conducted at the beginning of the study. This initial analysis resulted in first-order and second-order themes, as shown in Figure 5. These themes were derived directly from the GL codes and represent a higher level of abstraction and organization of the data. The strategies theme, while valuable for RQ1, was created later in the process and specifically for that question. They do not have the same range of data support as the original themes, which were derived from the comprehensive initial coding of all the GL documents. Using the original themes for the other research questions ensures that we're working with a consistent, well-supported set of concepts that span the entire dataset. Furthermore, the original themes offer a broader perspective that would be more appropriate for answering the other research questions, which might not be particularly strategy-focused but rather concentrate on additional aspects of the relationship between the development and production environments.

The theme **Data and Environment Management** deals with the need to ensure consistency of settings across contexts and to protect data integrity. The related code found is **Consistency of data and settings between environments** ensures that the data and configuration settings are kept consistent across different environments to avoid discrepancies and ensure smooth operation. For example, the document R8 mentioned: *"Habitat provides a uniform approach to building and deploying your application, with a final package that can run on bare metal servers or be exported to Docker and Kubernetes. This provides a consistent workflow that works for running your application on any platform."*.

The theme **Infrastructure Provisioning** focuses on the methods and resources needed to configure and oversee the infrastructure needed for software development and implementation, a process known as infrastructure provisioning. The first related code is **Automated Infrastructure Provisioning** utilizes automation tools and processes to provision and configure the required infrastructure resources for software development and deployment, streamlining the setup and management. For example, the document R81 declared: *"PaaS simply eliminates the cost and complexity of purchasing, configuring, and managing the hardware and software for developing applications. "*. The second related code is **Infrastructure Manual Provisioning** do not utilize automation tools and processes to provision and configure the required infrastructure resources for software development and deployment. For instance, the document R52 declared: *"Manual data handling seems cheap, but it does require a workforce, and it has a greater dependency on the professionals responsible. Admin and staff might need to work for hours, which might not be feasible as the business grows with time. It significantly increases staff hours which is neither efficient nor cost-effective."*. The third related code is **Reduced manual effort and errors** refers to a code associated to the automation of tasks and processes in order to mitigate the need for manual intervention and decrease the likelihood of human errors within the context of configuration management. For example, the document R87 mentioned: *'Developers can use IaC to create, manage, and modify infrastructure with a few lines of simple code. By using a PaaS that offers an easy IaC setup, developers can create and deploy applications faster and more reliably, while reducing the risk of error and ensuring that infrastructure is always consistent and up-to-date."*.

The theme **Monitoring and Performance** highlights the need of monitoring systems for tracking and assessing the functionality and condition of the application or system. The related code is **Monitoring Mechanisms** implements systems and tools to monitor the performance, availability, and health of software applications and infrastructure components. For example, the document R92 mentioned: *"Make sure you use monitoring tools that boost your visibility into the third-party service, so you don't waste your time fixing code if the problem lies with the server."*.

The theme **Team Collaboration and Communication** emphasis on effective teamwork and communication. The tasks assigned to each team must be clearly defined, cooperation must be seamless, and teams cannot be isolated from one another. The related code is **Time-saving**. This code refers to the perceived reduction in time and effort required to complete a task or achieve a goal. For example, the document R1 shared: *"Another best advantage of CI/CD is that it provides regular maintenance and updates of the product. It ensures that release cycles are short and targeted, which blocks fewer features that are not ready for release. In a CI/CD pipeline, maintenance is typically done during non-business hours, saving the entire team valuable time."*.

Finally, our study also reveals the theme **Validation and Quality Assurance** includes validation processes and technical debt management. The related code is **Build Validation Mechanisms** implement mechanisms to validate the integrity and correctness of software builds, ensuring that they meet quality standards and are suitable for deployment. For example, the document R16 mentioned: *"Profiles allow you to customize a particular build for a particular environment (development, testing, and production). Profiles facilitate portability between different build environments."*.

8 RQ4 – What are the perceived difficulties in the automation process in development and production environments?

Similarly to RQ3, we consider the GL themes from Figure 5 and the GL codes to answer this question.

The theme **Access Control and Security** manages and regulates access privileges to different settings. It is related to the code **Access restriction between environments** implementing measures to restrict access and control permissions between different environments (e.g., development, production) to maintain security and prevent unauthorized actions. For example, the document R4 mentioned: *"Development and testing teams often have access to limited resources or share an environment to test code changes. Sharing environments can be challenging for CD workflows."*.

The theme **Data and Environment Management** deals with the need to ensure consistency of settings across contexts and to protect data integrity. The first related code is **Dependencies between environments along the pipeline** identifies and manages dependencies between various environments within the software development and deployment pipeline to ensure smooth progression and avoid bottlenecks. For example, the document R8 declared: *"We must allow teams to construct the pipeline without worrying about the integrations involved."*. The second related code is **Inconsistency of data and settings between environments** highlights the presence of discrepancies and inconsistencies in data and configuration settings across different environments, which can lead to issues and errors. For example, the document R85 shared: *"PaaS brings forth a treasure trove of services and components that developers can cherry-pick to craft their applications. However, this abundant selection can inadvertently lead to the creation of application and data* *silos. Each service operates in its bubble, isolated from the rest, which can hamper communication and collaboration between components. This isolation often breeds data inconsistency and redundancy – a veritable headache for developers seeking to maintain a cohesive and streamlined application.”*. Another document R57 mentioned: *“Configuration items such as user accounts, compliance standards, and Information Security requirements often go unconfigured or misconfigured across operating systems, driving inconsistency and increasing risk.”*. Lastly, related code is **Lack of visibility into changes** indicates a challenge or issue related to the difficulty in obtaining clear and comprehensive visibility into alterations or modifications within a system or environment. This lack of visibility might hinder effective understanding, monitoring, and tracking of changes. For example, the document R48 declared: *“It is tremendously difficult to analyze the overwhelming amount of configuration information separating the critical from the non-critical to make sense of it all, and present it in a format that is precise and actionable. Essentially, how do you build IT analytics taking in account highly detailed change and configuration information?”*.

The theme **Infrastructure Provisioning** focuses on the methods needed to configure and oversee the infrastructure needed for software development and implementation, a process known as infrastructure provisioning. The related code is **Infrastructure Manual Provisioning**, related to the need of set up infrastructure resources (e.g., servers, databases) for software development and deployment. For example, the document R49 declared: *“One of the biggest drawbacks of using manual case management tools is the lack of flexibility and coordination they afford.”*.

The theme **Team Collaboration and Communication** emphasizes effective teamwork and communication. The tasks assigned to each team must be clearly defined, cooperation must be seamless, and teams cannot be isolated from one another. The first related code is **Team communication mechanisms**, concerning the processes and tools to facilitate effective collaboration among software teams. For example, the document R49 mentioned: *“When people use different workspace systems, collaboration becomes non-existent. Delegation of responsibilities can prove difficult to manage and ad-hoc tasks may involve last minute reshuffling of priorities and unwanted delays.”*. The second related code is **Well-defined scope of action between teams** clearly defining and communicating the areas of responsibility and authority for different teams involved in the software development and deployment process to avoid conflicts or duplication of efforts. For example, the document R49 mentioned: *“With manual tools, the required information and data exist in silos. And this makes it extremely difficult for case managers to access, contextualize, and derive actionable insights without having to refer to multiple sources and gather data.”*.

Finally, the theme **Validation and Quality Assurance** includes validation processes and technical debt management. The first related code is **Control of financial expenses** implements strategies to monitor and control the financial costs associated with software development and deployment activities. For example, the document R91 mentioned: *“Simply, all the server resources become unavailable until the problem is solved. Sometimes these downtimes can cost businesses millions of dollars.”*. The second related code is a **Technical debt** refers to the accumulated software development work that needs to be addressed or improved in the future due to shortcuts, temporary solutions, or suboptimal code. For example, the document R8 declared: *“However, insufficient expertise and inadequate training can result in several issues in the pipeline's implementation, such as incorrect process automation, flawed test case development, and faulty CI configuration.”*.

9 Discussion

9.1 Implications of our Findings

Our findings illustrate that, while effective when used independently, several strategies offer even greater value when combined to provide a more comprehensive approach to reducing variation in configuration between development and production environments. Teams may handle both high-level and granular parts of configuration management by combining strategies, ensuring that several considerations like consistency, security, and flexibility are taken into account simultaneously.

Our results also illustrate that strategies like the CMP and Automated Deployment Pipeline are effective when used independently, and also improve each other's efficacy when combined. For example, an automated deployment pipeline makes it straightforward for code to transition from development to production, and an organized configuration management plan ensures that every configuration can be tracked down and maintained properly. This combination can improve software system reliability by significantly reducing the possibility of discrepancies emerging during deployment.

Furthermore, the results of our study indicate that the theme of team collaboration and communication contributes to the successful implementation of the strategies. That is, executing strategies such as Secrets Management Platform and PaaS Deployment can be improved by implementing efficient communication mechanisms and clearly identifying roles within teams. For instance, teams who work well together and communicate well can quickly handle configuration changes and other problems, making the development process more responsive and agile. Combining these strategies and evaluating them together enables us to determine which combinations work best in certain circumstances. For example, companies that prioritize environment parity – making sure that the development and production settings are almost identical– in addition to diligent tracking and documenting procedures may find that configuration-related problems are mitigated. This alignment may result in more seamless software operations and effective solutions when inconsistencies arise.

9.2 General Recommendations

With these insights in mind, we advise practitioners to take a comprehensive approach to configuration management, utilizing the strategies that have been found to best suit their unique operating requirements. This can involve examining current procedures comprehensively and discovering how to incorporate or improve these strategies. Through this approach, companies may enhance their software delivery processes and cultivate a climate of creativity and constant development. This discussion also emphasizes the importance of continuing research in the configuration management field. Researchers must investigate new tools and strategies that handle new difficulties as the software development ecosystem changes. The knowledge acquired from this study can be used as a starting point for other research projects, promoting the investigation of different combinations of strategies and how they affect the techniques used to develop software. To improve their entire software development processes, practitioners are encouraged to take into account the interdependencies of the established strategies and to take a strategic approach to configuration management. By taking a holistic approach, companies are better prepared for current problems as well as future modifications in the quickly changing software market.

9.3 A Note About the High-level Themes

Data Governance and Management, and Operational Efficiency and Effectiveness, the two main subjects of our study, emerged due to the industry's increasing complexity in handling infrastructure configurations between development and production environments. The first theme emerged from the necessity to preserve consistency and protect sensitive information which gave rise to this theme. It is important to prevent divergence between development and production environments, particularly when working with numerous infrastructures. As cloud platforms, containers, and automation technologies like Infrastructure as Code (IaC) have been more widely used, it has become essential to guarantee the consistency, confidentiality, and integrity of configuration data across many contexts. The industry's push toward automation and scalability drives the second theme. The ability to smoothly move software from development to production without manual intervention becomes essential as deployment pipelines and infrastructure provisioning grow more automated. Organizations benefit from this efficiency through decreased operating costs, improved software stability in production settings, and quicker response times to changes.

10 Limitations

The generalizability and comprehensiveness of the study's findings may be affected by a number of recognized limitations. The main drawback is that we only used GL for our results, which covers publications like technical reports, and industry blogs. This method implies that we did not perform a Systematic Literature Review of peer-reviewed articles, even if it permitted us to obtain timely and pertinent perspectives from practitioners actively involved in the field. As a result, our conclusions about the differences between development and production environments might be biased and might not accurately reflect the current state of the art.

Dependence solely on GL could result in the exclusion of important techniques and ideas usually available in peer-reviewed literature, which is validated and examined with intense scrutiny. The lack of this information may lead to an inaccurate understanding of the difficulties and solutions associated with configuration management in software development. Additionally, it is possible that the search strategy adopted, which was restricted to the top 100 search engine results, provided extra bias and ignored pertinent websites.

However, we think the advantages of timeliness and relevance overcame these drawbacks for our particular study concerns and target audience. We employed several measures to mitigate potential biases and guarantee the integrity of our RR. We included all team members in the review process, followed well-defined inclusion and exclusion criteria, and employed an organized search process. Our RR searched for GL sources published during 2014 and 2024 to achieve this goal. Our search process identified 110 GL documents, that were analyzed in-depth using qualitative and quantitative approaches

Finally, we suggest that future studies use peer-reviewed materials and the Grey Literature to present a more comprehensive and unbiased analysis of the topic. By integrating insights gained from both forms of literature, researchers can cover a wider range of practices, challenges, and improvements within the field. This approach would improve the findings' validity and provide a more sophisticated comprehension of the challenges of handling configuration variations across development and production environments.

11 Related Works

Numerous earlier studies that empirically looked into the adoption, flaws, or difficulties of IaC connect to our work. Guerriero et al. (2019) provided light on the state of the practice in the adoption of IaC and the main software engineering difficulties in the sector by using data from 44 semi-structured interviews with senior developers of the same number of organizations. In particular, they discussed how IaC is developed and adopted by practitioners.

Jiang and Adams (2015) examined how infrastructure and production code co-evolved and found that there is a strong relationship between the two. As a result, when testers edit tests, they frequently alter the infrastructure specifications. Sharma and colleagues Sharma et al. (2016) examined configuration management tool source codes for code smells (such as Puppet and Chef). Consequently, they put up a list that included 11 design configuration smells and 13 implementation smells. After that, 4,621 open-source Puppet repositories were compared to the list. It's interesting to note that implementation smells were less common on average than design smells. In other words, a single poor or ineffective design choice leads to a host of subsequent quality problems.

Rahman et al. (2018) studied the difficulties in building IaC, particularly concerning configuration management tools. They searched Stack Overflow for frequently asked programming questions to assist IaC engineers. Once more, questions about puppets were given priority in this situation. The three most frequent question categories that they found using qualitative analysis were (i) syntax errors, (ii) provisioning instances, and (iii) assessing Puppet's feasibility of accomplishing certain tasks.

Shu et al. (2017) searched for security vulnerabilities in container images. They introduced a framework for automatically locating and analyzing images from the Docker HUB for Docker Image Vulnerability Analysis (DIVA). When all picture versions were taken into account, DIVA found an average of more than 180 vulnerabilities. Vulnerabilities might easily spread from parent to child images because many of the images had not even received updates in hundreds of days. More automated procedures for applying security updates to Docker images were suggested by the authors.

Ozdougan et al. (2023) the paper contributes to the growing body of research on IaC technologies. They extend previous studies by providing a comprehensive overview and classification of the most popular IaC tools, examining their key characteristics, and comparing them across various dimensions. Their findings offer valuable insights for practitioners and researchers seeking to understand and leverage IaC technologies effectively.

Hasan et al. (2020) the study contributes to the growing body of research on IaC testing. They extend previous studies by conducting a comprehensive review of existing literature and identifying six key testing practices for IaC scripts. Their findings provide practitioners and researchers with a valuable resource for improving the quality and reliability of IaC deployments.

Lastly, Rahman et al. (2021) the work builds upon existing research on secure IaC development. They extend previous studies by conducting a comprehensive grey literature review to identify 12 practices for secret management in IaC, including both general and language-specific approaches. Their findings offer practitioners and researchers valuable insights into securing secrets in IaC environments.

These related works discuss earlier studies that have looked into the adoption, flaws, or difficulties of IaC. In contrast, this work focuses specifically on strategies to mitigate configuration differences in software development, using an RR of GL as the research method. Our study is unique in that aims to answer four research questions related to the actions taken by software teams to mitigate differences between development and production environments, the efficacy of these actions, the benefits of automation, and the difficulties in the automation process.

12 Conclusions

This study aimed to identify efficient strategies to manage configuration differences between development and production environments in software development. We acquired information from multiple documents through a Rapid Review of Grey Literature, which enabled us to identify nine important strategies companies may use to mitigate these differences including the Configuration Management Plan, Automated Deployment Pipeline, and others. Our findings suggest that these strategies strengthen the overall efficiency and consistency of software systems in addition to helping reduce configuration-related challenges.

We observed that the strategies we identified are important to bridge the gap between development and production environments. In particular, we found that the adoption of automation tools and efficient teamwork and communication are fundamental to the successful execution of these strategies. For example, the Automated Deployment Pipeline reduces the possibility of errors that can occur from human operations by facilitating seamless transitions from development to production. In addition, our analysis showed that different people had distinct perspectives on how sufficient various strategies were, suggesting that while some are considered enough, others could need to be improved upon or given further support to be completely effective.

This study's insights highlight how crucial it is to comprehend the complexity of configuration management. Software quality and time to market can be negatively impacted by the inherent conflict between the agility development teams desire and the stability that operation teams require. Organizations can optimize software delivery outcomes by fostering a more collaborative atmosphere that aligns both teams' intentions through the strategies suggested in this research.

Although this study offers insightful information, it also emphasizes the need for more research. To confirm and build on our findings, a conventional systematic literature review (SLR) on this subject would be helpful. Peer-reviewed literature included in such an SLR can provide an improved understanding of the state of the art in handling configuration differences. This would strengthen the validity of the results and deliver the strategies with a more comprehensive framework, enabling a more sophisticated discussion of how they may be applied in various organizational contexts.

In conclusion, this study promotes the topic by offering practical strategies for addressing configuration management issues in software development and deploymen. Our objective in concentrating on the insights found in Grey Literature is to provide practitioners with practical advice that they can use immediately in real-world situations. The results of this study encourage both researchers and practitioners to delve deeper into the intricacies of configuration management by providing a basis for future research on this subject. We support continued discussion and study in this field to continue improving software development best practices and eventually produce more dependable and efficient software systems.

In our next work, we intend to conduct an RR over the GL to find the key factors influencing the decision-making process when selecting strategies to mitigate configuration differences in software development. Altogether, the main contributions of this future research will be:

- It offers an understanding of the key factors that affect the selection of mitigation strategies for configuration differences. Thanks to this knowledge organizations are better equipped to make decisions that are in line with their unique requirements and circumstances;
- Software maintenance requires dealing with technical debt brought on by configuration changes. The study helps companies decide how to reduce technical debt and stop it from accumulating;
- For a seamless user experience, development and pro-

duction environments must be consistent. The study emphasizes the significance of user experience in selecting a strategy by guaranteeing consistent functionality; and
- It emphasizes the significance of selecting long-term sustainable strategies, ensuring configurations are still relevant and adaptable as technology advances.

The implications of this future research can be different for industry and academia. For industry, the practitioners can benefit from a better understanding of the key factors influencing strategy selection. This information enables businesses to select configuration strategies for management with greater awareness, which results in more successful and efficient software development processes. The companies can optimize their configuration management processes, reducing inefficiencies and minimizing risks related to configuration differences by taking into account the research's components. Software deployments may go more smoothly as a result, and system stability may also increase. They can also modify their processes to align with their chosen approach, whether Agile, DevOps, or another methodology, by understanding how different development methods affect strategy choices. Finally, for businesses looking to reduce technical debt accumulation, which can increase development costs and reduce system maintainability, the study's findings on managing technical debt associated with configuration modifications may be helpful.

For academia, in software engineering and configuration management, the research adds to the body of knowledge. It offers a framework for additional investigation into the precise variables impacting strategy choice and its effects. Academics can use the research findings to create teaching tools and curriculum materials for software engineering and IT management courses. It helps students comprehend the complexity of configuration management decision-making. It promotes interdisciplinary cooperation between software engineering and other disciplines, including risk management, regulatory compliance, and cloud computing. This multidisciplinary approach might result in a more thorough comprehension of configuration management. Finally, a theoretical framework and best practices for configuration management can be developed as a result of academic research that can validate and improve the findings.

13 Data Availability.

The data for this work, including the collected data with the quality assessment checklist, analyzed data, codes, and employed query strings, is publicly available at `https://doi.org/10.5281/zenodo.10530032`.

Acknowledgements

The authors would thank the reviewers for their helpful comments. This work was partially supported by CNPq (grants 308623/2022-3 and 314834/2021-4) and FAPESPA (053/2021).

References

Adams, J., Hillier-Brown, F. C., Moore, H. J., Lake, A. A., Araujo-Soares, V., White, M., and Summerbell, C. (2016). Searching and synthesising 'grey literature'and 'grey information'in public health: critical reflections on three case studies. *Systematic reviews*, 5(1):1–11.

Adams, R. J., Smart, P., and Huff, A. S. (2017). Shades of grey: guidelines for working with the grey literature in systematic reviews for management and organizational studies. *International Journal of Management Reviews*, 19(4):432–454.

Cartaxo, B., Pinto, G., and Soares, S. (2018). The role of rapid reviews in supporting decision-making in software engineering practice. In *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018*, pages 24–34, New York, NY, United States. Association for Computing Machinery.

Cruzes, D. S. and Dyba, T. (2011). Recommended steps for thematic synthesis in software engineering. In *2011 international symposium on empirical software engineering and measurement*, pages 275–284, none. IEEE, none.

Ebert, C., Gallardo, G., Hernantes, J., and Serrano, N. (2016). Devops. *Ieee Software*, 33(3):94–100.

Fitzgerald, B. and Stol, K.-J. (2017). Continuous software engineering: A roadmap and agenda. *Journal of Systems and Software*, 123:176–189.

Garousi, V. and Felderer, M. (2017). Experience-based guidelines for effective and efficient data extraction in systematic reviews in software engineering. In *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*, EASE '17, page 170–179, New York, NY, USA. Association for Computing Machinery.

Garousi, V., Felderer, M., and Mantyla, M. V. (2019). Guidelines for including grey literature and conducting multivocal literature reviews in software engineering. *Information and software technology*, 106:101–121.

Guba, E. G. (1981). Eric/ectj annual review paper: Criteria for assessing the trustworthiness of naturalistic inquiries. *Educational Communication and Technology*, 29(2):75–91.

Guerriero, M., Garriga, M., Tamburri, D. A., and Palomba, F. (2019). Adoption, support, and challenges of infrastructure-as-code: Insights from industry. In *ICSME*, pages 580–589, None. IEEE.

Hasan, M. M., Bhuiyan, F. A., and Rahman, A. (2020). Testing practices for infrastructure as code. In *Proceedings of the 1st ACM SIGSOFT International Workshop on Languages and Tools for Next-Generation Testing*, LANGETI 2020, page 7–12, New York, NY, USA. Association for Computing Machinery.

Jiang, Y. and Adams, B. (2015). Co-evolution of infrastructure and source code - an empirical study. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, pages 45–55, New York City at 3 Park Ave. IEEE.

Kamei, F., Wiese, I., Lima, C., Polato, I., Nepomuceno, V., Ferreira, W., Ribeiro, M., Pena, C., Cartaxo, B., Pinto, G., and Soares, S. (2021). Grey literature in software engineering: A critical review. *Information and Software Technology*, page 106609.

Kamei, F., Wiese, I., Pinto, G., Ribeiro, M., and Soares, S. (2020). On the use of grey literature: A survey with the brazilian software engineering research community. In *Proceedings of the 34th Brazilian Symposium on Software Engineering*, SBES '20, pages 183–192, New York, NY, USA. Association for Computing Machinery.

Leite, L., Rocha, C., Kon, F., Milojicic, D., and Meirelles, P. (2019). A survey of devops concepts and challenges. *ACM Comput. Surv.*, 52(6).

Mazyar, M. (2018). Devops: The ultimate way to break down silos - devops.com. `https://devops.com/devops-the-ultimate-way-to-break-down-silos/`. (Accessed on 01/27/2023).

Monclus, P. (2021). Multi-cloud connectivity and security needs of kubernetes applications. `https://blogs.vmware.com/networkvirtualization/2021/05/multi-cloud-connectivity-security-kubernetes.html/`. (Accessed on 08/18/2023).

Nazário, M., Bonifácio, R., de Souza, C. R. B., Kenji, F., and Pinto, G. (2024). Strategies to mitigate differences between environments using rapid review on grey literature.

Ogawa, R. T. and Malen, B. (1991). Towards rigor in reviews of multivocal literatures: Applying the exploratory case study method. *Review of educational research*, 61(3):265–286.

Ozdougan, E., Ceran, O., and Ustundaug, M. T. (2023). Systematic analysis of infrastructure as code technologies. *Gazi University Journal of Science Part A: Engineering and Innovation*, pages 452–471.

Patro, S. and Sahu, K. K. (2015). Normalization: A preprocessing stage. *arXiv preprint arXiv:1503.06462*, 1(1):1.

Rahman, A., Barsha, F. L., and Morrison, P. (2021). Shhh!: 12 practices for secret management in infrastructure as code. In *2021 IEEE Secure Development Conference (SecDev)*, pages 56–62.

Rahman, A., Partho, A., Morrison, P., and Williams, L. (2018). What questions do programmers ask about configuration as code? In *2018 IEEE/ACM 4th International Workshop on Rapid Continuous Software Engineering (RCoSE)*, pages 16–22, New York City at 3 Park Ave. IEEE.

Sharma, T., Fragkoulis, M., and Spinellis, D. (2016). Does your configuration code smell? In *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*, pages 189–200, New York City at 3 Park Ave. IEEE.

Shu, R., Gu, X., and Enck, W. (2017). A study of security vulnerabilities on docker hub. In *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, CODASPY '17, page 269–280, New York, NY, USA. Association for Computing Machinery.

Tichy, W. F. (1995). *Configuration management*. John Wiley & Sons, Inc., 605 Third Ave. New York NY 10158 USA.

Wohlin, C. (2014). Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th international conference on evaluation and assessment in software engineering*, pages 1–10, New York, NY, United States. Association for Computing Machinery.