The Journal of Systems and Software 235 (2026) 112761

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

Mitigation strategies for confidentiality violations in software architecture using ranked feature importance

Nils Niehues[a,*], Sebastian Hahner[a], Robert Heinrich[b]

[a] KASTEL – Institute of Information Security and Dependability, Karlsruhe Institute of Technology (KIT), Am Fasanengarten 5, 76131, Karlsruhe, Germany
[b] Institute of Software Engineering and Programming Languages, Ulm University, Albert-Einstein-Allee 11, 89069, Ulm, Germany

ARTICLE INFO

Editor: Prof Raffaela Mirandola

Keywords:
Software architecture
Confidentiality
Uncertainty
Machine learning
Data flow analysis

ABSTRACT

A quality attribute like confidentiality is critical to trustworthy software but unfortunately, very challenging to ensure. This is because modern software systems are complex and interconnected. Architecture-based confidentiality analysis enables the early detection of violations, helping to mitigate risks before deployment. However, uncertainty in software systems and their environments complicates precise and comprehensive architectural analysis. Additionally, the complexity of software models and the exponential growth of uncertainty scenarios pose significant challenges for automated mitigation, often leaving software architects to resolve confidentiality violations manually, a process that is both time-intensive and error-prone.

In this paper, we extend our machine-learning-based approach to mitigate confidentiality violations. Specifically, we introduce a novel mitigation strategy inspired by TCP Congestion Control, as well as a strategy that capitalizes on clustering techniques to dynamically adjust batch sizes. Our evaluation on three real-world software architectures demonstrates that our extended approach can mitigate confidentiality violations while outperforming the state-of-the-art. Whereas previously the upper limit was 60 times runtime reduction, now we achieve 2298 times reduction, with the median being an elevenfold reduction. Our statistical analysis confirms that the added TCP-inspired strategy is significantly cheaper than the state-of-the-art baseline (Friedman test $p = .025$ and Nemenyi post hoc test $p = .039$), while also having a strong practical impact (Kendall's $W = 0.721$). This extended work deepens our understanding of the nature of uncertainty and also of the techniques optimally suited to mitigating the violations caused by uncertainties. It takes us one step closer to designing trustworthier systems.

1. Introduction

In today's digital landscape, software systems are becoming increasingly interconnected and more complex, necessitating early and thorough analysis to ensure security-critical quality attributes, such as authenticity, integrity, and confidentiality. Confidentiality is especially associated with a sense of trust in software. Confidentiality guarantees that data is "accessible and interpretable only by authorized users within a specific context of use" (ISO/IEC, 2018). Moreover, the ISO/IEC defines trustworthiness (ISO/IEC, 2011) as a combination of security, safety, reliability, resilience, and privacy. Confidentiality is a critical part of security, and as such, goes a long way to making software trustworthy. Therefore, quality issues like confidentiality violations must be identified and also mitigated. However, maintaining confidentiality is increasingly put at risk by such erratic and unpredictable factors as changes in user behavior or in the system environment, both of which may negate the initial assumptions and conditions under which confidentiality measures were designed (Lehman and Fernáandez-Ramil, 2006; Hahner, 2025). That unknowable deviation between design behavior and actual behavior is what we call *uncertainty*, much as do, too, researchers in self-adaptive systems, where uncertainty is defined as "any deviation of deterministic knowledge that may reduce the confidence of adaptation decisions made based on the knowledge" (Weyns, 2020).

The proactive approach of architecture-based confidentiality analysis enables software architects to detect potential confidentiality violations before they occur in the deployed system. By reliably and automatically identifying issues early in the development process, architecture-based confidentiality analysis ensures not just a level of trustworthiness but crucially as well, a necessary degree of cost-efficiency, meaning, when issues are addressed earlier in the development cycle, costs are lowered dramatically (Glinz, 2006; Boehm and Basili, 2001). Data flow and architectural diagrams assist in the early

* Corresponding author.
E-mail addresses: nils.niehues@kit.edu (N. Niehues), sebastian.hahner@kit.edu (S. Hahner), robert.heinrich@uni-ulm.de (R. Heinrich).

https://doi.org/10.1016/j.jss.2025.112761
Received 3 March 2025; Received in revised form 28 November 2025; Accepted 22 December 2025
Available online 26 December 2025

identification of sources of uncertainty which may affect confidentiality (Schneider et al., 2024). Importantly, most sources of uncertainty can be addressed during the design phase (Hahner et al., 2023). For example, uncertainties about server deployment locations or about encryption activation can influence data flow and compliance with confidentiality requirements. Overall, analysis approaches based on software architecture effectively identify and express sources of uncertainty.

Notwithstanding, a significant challenge remains in assessing the impact of uncertainty on confidentiality requirements, and this assessment is rendered all the more challenging by the complexity of modern software systems and the numerous forms of uncertainty thus entailed (Hahner, 2021). For instance, if it is uncertain where a server processing personal data is located, that may cause potential regulatory violations (e.g., under the GDPR (Union, 2016)). Moreover, multiple uncertainty sources can interact (Camara et al., 2024), resulting in previously unpredicted uncertainty interactions. For example, missing encryption might cause a violation only if the database server is deployed outside of the European Union. Hence, it is insufficient to identify sources of uncertainties or analyze their impact merely in isolation.

One attempt to analyze such quality attributes as confidentiality under uncertainty is the *Architecture-Based and Uncertainty-Aware Confidentiality Analysis (ABUNAI)* approach (Hahner, 2025). The *ABUNAI* approach (Hahner, 2025) provides a viable solution because it detects violations of confidentiality constraints that are caused, ultimately, by uncertainties existing in any relation to the system at all. *ABUNAI* can effectively identify confidentiality violations, even where multiple uncertainty sources interact. Essentially, by expressing different scenario combinations as independent data flows, *ABUNAI* operationalizes uncertainty so as to achieve maximal coverage of sources of uncertainty. The limitation of *ABUNAI*, however, is that it does not offer solutions to mitigate those violations (Seifermann et al., 2022; Peldszus et al., 2019; Hahner, 2025). Consequently, software architects must resort to manually resolving each identified confidentiality violation, a process that is both labor-intensive and prone to human error, especially in large-scale systems (Xu and Zhou, 2015). Additionally, the exponential growth of possible combinations of uncertainty scenarios presents a major challenge for automated solutions. Exploring all possible configurations quickly becomes computationally prohibitive, a known issue in the field of design space exploration (Koziolek et al., 2011; Koziolek, 2011).

Our previous work (Niehues et al., 2025) addressed these shortcomings and mitigated confidentiality violations using machine learning that (a) assessed uncertainty impact by a ranking algorithm and that (b) prevented violations by an automated mitigation mechanism that only focuses on the relevant uncertainties from (a). That approach enabled the effective maintenance of confidentiality requirements while also reducing analysis runtime by a factor of 60 compared to the state-of-the-art method (Hahner, 2025). Nonetheless, our previous approach incurred considerable cost in runtime latencies, and worse, it disregarded the importance of uncertainty dynamics as well as the rapidity with which even small but redundant combinations will escalate the mitigation runtime.

In that previous work (Niehues et al., 2025), we employed fixed batch sizes to divide uncertainties into equal parts so that we could test all possible combinations within each batch. The method was definitely straightforward, but it failed to account for unique structural variations in the models, where dynamically adjusting batch sizes can yield better results. Also, as to the cost-inefficiency of our previous approach, we required unnecessary checks at the start of the process because we failed to appreciate how models often contain numerous uncertainties, several of which contribute to confidentiality violations. Consequently, when we tested scenarios where the number of uncertainties was insufficient, our approach was unable to consider all relevant uncertainties. Taken together, these shortcomings of our previous approach point the way forward to a comprehensive architecture-based analysis that leverages machine learning to efficiently mitigating violations in confidentiality.

The novel contributions of our extension here of Niehues et al. (2025) are as follows:

C1 Our extended approach adapts techniques from telematics to tackle the plethora of unnecessary combinations that impede the initialization of mitigation. To that end, we introduce a first new mitigation strategy inspired by TCP Congestion Control (Blanton et al., 2009). In TCP Congestion Control, window sizes grow exponentially until a threshold is reached, and we have adopted this same technique to enable faster mitigation of all confidentiality violations caused by uncertainties.

C2 Our extended approach adapts batch sizing dynamically and so is able to ensure a more coherent and effective mitigation process. To that end, we introduce a second new mitigation strategy. It uses clustering techniques — as opposed to arbitrary fixed slices — to adjust batch sizes based on the input model dynamically.

C3 An enhanced evaluation provides statistical significance for the state-of-the-art performance of our extended approach. We found a significant difference (Friedman test $p = .025$) between our strategies and the baseline and as well, a strong practical impact (Kendall's W $= 0.721$). Further, our new strategy in **C1** proved to be significantly faster than the baseline (Nimenyi post hoc test $p = .039$).

The paper is organized as follows. Section 2 describes the foundations for this work and Section 3 introduces a running example. In Section 4, we provide an overview of our extended approach, and this sets the stage for the uncertainty ranking in Section 5 and for the mitigation of violations in Section 6. We evaluate our work in Section 7, and discuss the results and broader implications in Section 8. Finally, we present related work in Section 9 and conclude in Section 10.

2. Foundations

This section outlines the foundations essential for understanding the contributions of this work. It covers the syntax of Data Flow Diagrams (DFDs), its application in confidentiality analysis, the role of uncertainty in software architecture, and the machine learning techniques employed to address constraint violations in uncertain systems.

2.1. DFDs and confidentiality analysis

DeMarco (1978) introduced DFDs to visually represent how data moves through a system, including its processing steps. A network of nodes and flows represents the data movement through a system, where nodes correspond to entities like processes, data stores, or external entities. System analysts widely use DFDs because they clearly and simply describe how a system processes data using a network of data transformers (Larsen et al., 1994; Schneider et al., 2024).

The classic DFD model lacks the expressiveness required for advanced applications like detecting data flow constraint violations or addressing non-functional attributes such as access control and hardware constraints. To address this, Seifermann et al. (2022) introduced an improved DFD metamodel with structures for security properties and resource allocations, enabling automatic constraint checks and greater system complexity representation.

To address rule explosion and performance bottlenecks with large systems, Boltz et al. (2023) developed an improved DFD-based model. Their method leverages Transposed Flow Graphs (TFGs), which are directed and acyclic graphs that trace the flow of data through the system from many sources to one sink enabling efficient constraint checking. This model includes the following key elements:

Nodes are categorized into external nodes (data sources/sinks), process nodes (which modify and forward data), and store nodes (which store and emit data). *Flows* represent data transfer between nodes. *Labels* are used to annotate node data characteristics (e.g. *Encrypted* or *Personal*), which propagate through the DFD and are critical for assessing confidentiality. *Behaviors* define how data is processed at each node, with assignments and pins that specify input/output operations. Assignments forward data or modify properties by adding or removing labels.

By augmenting traditional DFDs with these features, the Data Flow Analysis (DFA) can check complex requirements such as access control, geographical data restrictions, and hardware-related constraints and detect violations (Boltz et al., 2023). As such an analysis either finds confidentiality violations or does not, we consider confidentiality a strictly binary attribute.

2.2. *Uncertainty in software architecture*

Uncertainty is defined as “any deviation of deterministic knowledge that may reduce the confidence of adaptation decisions made based on the knowledge” (Weyns, 2020). Uncertainty in software systems arises from incomplete or ambiguous information about the system’s design or operation and can impact various aspects of system behavior, including security, performance, and reliability (Walker et al., 2003). In the context of this work, we address uncertainties in software architecture models and their influence on confidentiality. Here, we focus on *software-architectural uncertainty* that resolves the lastest during the deployment and can effectively be addressed using architectural approaches (Hahner et al., 2023).

Hahner et al. (2023) proposed a classification of architectural uncertainty that can be applied to DFDs to model the impact of uncertainty on data flow and constraint violations. The classification identifies different types of uncertainties, each related to architectural elements in the DFD: *Behavior uncertainties* relate to the specific actions or functions process nodes perform. For example, there may be uncertainty regarding whether a node will apply encryption or simply forward the incoming data. *Node uncertainties* describe the properties of external nodes. For instance, it may be uncertain whether a database server is deployed within or outside the European Union. *Flow uncertainties* affect the possible routes that data may take through the system, including potential variations in the data pathway. For example, there may be uncertainty about whether data will bypass certain nodes or flow through additional processing stages.

Hahner (2025) models these uncertainties in DFDs using the Architecture-Based and Uncertainty-Aware Confidentiality Analysis (ABUNAI) approach, which analyzes combinations of uncertainty scenarios to identify confidentiality violations. This helps to identify risks and violations earlier in the design process and providing the fundamentals for this work.

2.3. *Machine learning techniques*

This work employs different supervised and unsupervised machine learning techniques to analyze the impact of different uncertainties on data flows and identify patterns that could lead to confidentiality violations. The goal is to detect violations and understand their underlying causes, helping to prioritize uncertainties for resolution. This section provides a short overview of the utilized techniques. Exploratory Factor Analysis (EFA) simplifies complex datasets by uncovering latent factors underlying variable relationships (Hooper, 2012; Yong et al., 2013). Principal Component Analysis (PCA) reduces dimensionality by transforming variables into orthogonal components, revealing primary variance sources (Abdi and Williams, 2010). Random Forests (RF) improves predictive accuracy through an ensemble of decision trees, especially in high-dimensional data (Breiman, 2001). Linear Regression (LR) and Logistic Regression (LGR) model relationships between inputs and outcomes, offering interpretable coefficients and binary classification, respectively (Su et al., 2012; Nick and Campbell, 2007). Linear Discriminant Analysis (LDA) enhances class separation by projecting data into lower-dimensional space (Balakrishnama and Ganapathiraju, 1998).

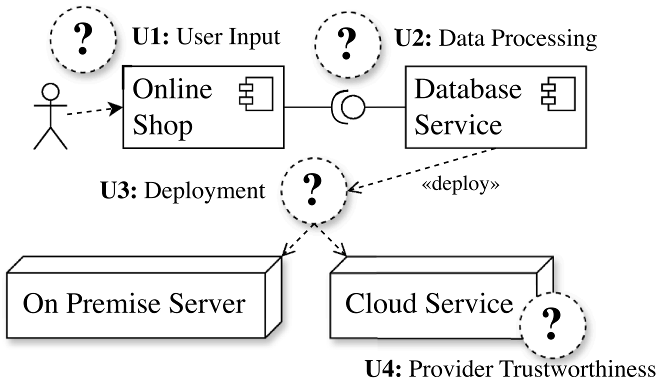


Fig. 1. Running example of a simple online shop.

3. Running example

We illustrate the application and impact of our mitigation approach with a simplified model of an online shop. Fig. 1 shows the architecture model consisting of the user, two software components, and two data storages and has been used in similar work (Hahner et al., 2024). The *User* sends data to the *Online Shop* component, which directs it to the *Database Service*. The *Database Service* stores data on either the *Cloud Service* or the *On-Premise Server*. We identified four uncertainties in this online shop, depicted by dotted cycles with a question mark.

U1 The user input may contain either public or personal data
U2 The data might get sanitized or forwarded unfiltered
U3 It is unclear on which server the data will be stored
U4 The location of the cloud server is unclear, e.g., inside or outside the European Union.

We define two confidentiality requirements for our model: personal data shall never leave the European Union and shall always be sanitized. Violations of these requirements could lead to data breaches or regulatory non-compliance.

We use the predicates $hasDataLabel$ and $hasNodeLabel$ to tag each data flow and component with specific characteristics, verifying compliance with confidentiality requirements: Using these predicates, we can formalize our constraints:

$$hasDataLabel(Personal) \wedge \neg hasNodeLabel(nonEU) \tag{1}$$

$$hasDataLabel(Personal) \wedge hasDataLabel(Sanizited) \tag{2}$$

Depending on the configuration of our model, both are violated. The first is violated if the user inputs personal data that flows to the cloud server with a non-EU location. The second is violated if the user inputs personal data that is not sanitized at the processing node

To automatically modify the model without violating constraints, one could try all combinations of uncertainty values and check for constraint violations in each resulting model.

As displayed in Table 1, this leads to an exponentially increasing number of configurations that need to be checked, with two options for four uncertainties sources being $2^4 = 16$. We found a solution if one of these configurations does not violate the given constraints.

When looking at the table, one might notice that certain uncertainties, such as the nature of user input, have a more significant impact on the outcome. For example, if the input is public, a violation is avoided regardless of other settings. Therefore it makes sense to try variations of the important uncertainties first to reduce the complexity and runtime.

In complex models with multiple uncertainties, prioritizing them manually is impractical, tedious, and error-prone, which motivates our

Table 1
All possible combinations for the given uncertainties.

Input	Processing	Deployment	Location	Violation
Personal	Sanitized	On Premise	EU	False
Personal	Sanitized	On Premise	non EU	False
Personal	Sanitized	Cloud	EU	False
Personal	Sanitized	Cloud	non EU	True
Personal	Unfiltered	On Premise	EU	True
Personal	Unfiltered	On Premise	non EU	True
Personal	Unfiltered	Cloud	EU	True
Personal	Unfiltered	Cloud	non EU	True
Public	Sanitized	On Premise	EU	False
Public	Sanitized	On Premise	non EU	False
Public	Sanitized	Cloud	EU	False
Public	Sanitized	Cloud	non EU	False
Public	Unfiltered	On Premise	EU	False
Public	Unfiltered	On Premise	non EU	False
Public	Unfiltered	Cloud	EU	False
Public	Unfiltered	Cloud	non EU	False

machine-learning approach to efficiently identify important uncertainties based on existing violations, making it suitable for scalable applications in complex models. While checking 16 combinations for 4 uncertainties by hand may be doable, checking 1024 for 10 is not.

4. An approach to rank and mitigate confidentiality violations

This paper extends our previous approach (Niehues et al., 2025) for analyzing and adjusting DFDs to ensure compliance with confidentiality constraints while also accounting for uncertainties in the model. In this section, we provide an overview of our extended approach. As illustrated in Fig. 2, our extended approach focuses on identifying and addressing uncertainties that could lead to confidentiality breaches. Our approach systematically detects and configures valid scenarios within the input model. By modifying only the necessary uncertainties to achieve compliance, we maintain the integrity of the original model as much as possible while also optimizing the analysis runtime.

In the first part of our approach, we pinpoint uncertainties that cause constraint violations. To achieve this, we implement a machine-learning ranking system that distinguishes relevant uncertainties from those that do not affect confidentiality. We begin the approach by transforming the output of an uncertainty-aware confidentiality analysis into categorical training data, which highlights the effects of various modeled scenarios on constraint violations. We then apply a range of machine learning techniques—both unsupervised and supervised—including EFA, PCA, RF, LR, LGR and LDA, to rank uncertainties based on their potential to cause violations. This ranking helps prioritize uncertainties in subsequent steps.

In the second part of our approach, we iteratively mitigate these ranked uncertainties by systematically testing combinations of scenarios to resolve constraint violations while minimizing disruption to the model. We use a greedy approach to apply various mitigation strategies, ranging from modifying single uncertainties to testing broader subsets, to find configurations that best satisfy confidentiality constraints. This iterative refinement ensures a balance between runtime efficiency and minimal impact on the original model structure, adjusting only the most critical uncertainties for achieving compliance.

In cases like the running example where multiple solutions are possible, the approach further optimizes for solutions that modify the fewest number of uncertainties. By exploring alternative configurations that leave as many uncertainties as possible intact, we aim to retain the flexibility of the original DFD, while complying with confidentiality constraints.

This combined method offers an advancement by enabling automated, efficient mitigation of confidentiality violations within uncertain DFDs.

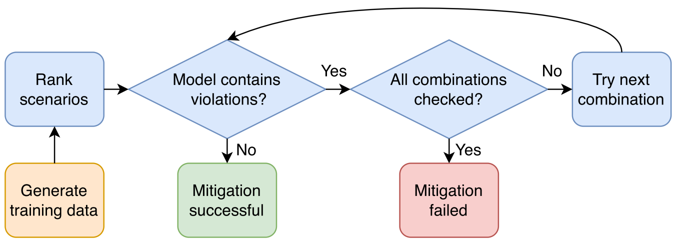


Fig. 2. Simplified pipeline of the approach, where blue denotes automatic activities and orange denotes semi-automatic activities. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

5. Ranking uncertainties based on constraint violations

In our previous work (Niehues et al., 2025), we introduced an approach for ranking uncertainties by analyzing their impact on constraint violations using ranked feature importance. This approach begins with methods for generating training data tailored for ranking purposes. Next, we apply techniques to assess the importance of uncertainties, followed by a strategy to aggregate rankings across multiple constraints. Each step leverages machine learning to systematically prioritize uncertainties with the greatest potential impact on confidentiality or other constraints.

5.1. Generating training data for ranking uncertainties

To begin, we use Transposed Flow Graphs (TFGs) to generate training data for uncertainty ranking. Each TFG represents a unique combination of uncertainty scenarios that is checked by ABUNAI to determine whether a confidentiality violation occurs (Hahner, 2025). For each constraint here, we generate separate training data automatically. Thus, the uncertainties themselves do need to be specified manually, but the training-data generation is fully automated via ABUNAI. For example, only 4 uncertainties require specifying in our running example, and the $2^4 = 16$ TFGs are generated automatically by ABUNAI. Next, each TFG as well as its violation status are automatically translated into categorical data which are input-ready for the machine learning techniques. Each row corresponds to a TFG, while each column corresponds to a specific uncertainty scenario. To maintain a consistent tabular format, we use placeholder values for missing uncertainties wherever a TFG does not flow through all uncertainty sources. This structured approach allows for machine learning models to process data even when some uncertainties are absent in specific scenarios. To balance the dataset, we include both TFGs that cause confidentiality violations and TFGs that do not. This improves the model's ability to identify critical uncertainties by reducing the noise from irrelevant configurations.

Table 2 presents the categorical training data derived from transforming TFGs of our running example (Section 3). Since the cloud server's location is irrelevant if we store data on the on-premise server, these scenarios are marked with -. The example consists of 4 uncertainty sources, each with 2 scenarios, and we check both constraints on each scenario combination. These results are shown in columns C1 and C2 of Table 2.

5.2. Ranking uncertainties for importance

Using machine learning, we can prioritize uncertainties by analyzing how their different scenarios influence constraint violations. To derive rankings, we employ both unsupervised and supervised learning approaches.

Unsupervised Approaches rely on identifying underlying patterns in TFG-based training data. Techniques such as PCA and EFA are useful for reducing data complexity while still retaining the most critical

Table 2
Training data for Fig. 1, where **U** stands for uncertainty, **C** for constraint, **S** for scenario and **-** for irrelevant.

U1	U2	U3	U4	C1	C2
S1	S1	S1	–	False	False
S1	S1	S1	–	False	False
S1	S1	S2	S1	False	False
S1	S1	S2	S2	True	False
S1	S2	S1	–	False	True
S1	S2	S1	–	False	True
S1	S2	S2	S1	False	True
S1	S2	S2	S2	True	True
S2	S1	S1	–	False	False
S2	S1	S1	–	False	False
S2	S1	S2	S1	False	False
S2	S1	S2	S2	False	False
S2	S2	S1	–	False	False
S2	S2	S1	–	False	False
S2	S2	S2	S1	False	False
S2	S2	S2	S2	False	False

variables (Abdi and Williams, 2010; Williams et al., 2010). PCA prioritizes uncertainties, which explain the highest variance in the data, generating components emphasizing highly correlated variables. EFA, on the other hand, focuses on latent factors that reveal relationships between uncertainties. Both approaches rank uncertainties by the degree to which they correlate with key data patterns, derived from component or factor loadings (as outlined in Algorithm 1). This method is effective for uncovering those subtle patterns across TFGs which might influence constraint violations.

Algorithm 1 Ranking unsupervised.
```
function RankUnsupervised(data_rows)
    ranking ← {}
    result ← CreateComponents(data_rows)
    components ← result.Components
    loadings ← components.Loadings
    for $loading \in loadings$ do
        continue for Constraint violated column
        ranking[loading.Name] ← sum(loading.Values)
    end for
    return SortDictDescending(ranking)
end function
```

Supervised Approaches provide direct insight into the relationship between uncertainties and violations by assessing confidentiality violations in different scenarios. Techniques such as LDA, RF, LR, LGR are applied to classify TFGs according to violation status (Linardatos et al., 2020). Each technique assesses the importance of an uncertainty based on its impact on the prediction outcome. For instance, LDA calculates linear discriminants that separate violation and non-violation cases, with coefficients indicating the influence of each uncertainty (Balakrishnama and Ganapathiraju, 1998). In contrast, Random Forests measure variable importance through mean decreases in impurity across decision trees, capturing both individual and combined effects of uncertainties (imp, 2024; Breiman, 2001). As illustrated in Algorithm 2, this approach for deriving feature importance from each model provides a foundation for ranking uncertainties by their contribution to constraint violations.

To illustrate the ranking result, we use logistic regression to rank the training data for the running example shown in Table 2. Table 3 shows the importance of uncertainties for the individual constraints. In Table 3a, **U4** is the most important uncertainty for constraint 1, which makes sense since deploying the cloud server inside the EU will never lead to a violation. Similarly, Table 3b shows that **U1** and **U2** are equally important for constraint 2 because only combinations of these two can lead to a sanitization violation.

Algorithm 2 Ranking supervised.
```
1: function RankSupervised(data_rows)
2:     model ← SupervisedMachineLearningTechnique()
3:     fitModel(model, data_rows)
4:     feature_imporances ← model.Feature_importances
5:     feature_names ← data_rows.Column_names
6:     return {feature_names, feature_imporances}
7: end function
```

Table 3
Individual importance of uncertainties for the running example.

Uncertainty	Importance
(a) Constraint 1.	
U4	0.6307
U1	0.5836
U3	0.5713
U2	0.5000
(b) Constraint 2.	
U1	0.6250
U2	0.6250
U3	0.5000
U4	0.5000

5.3. *Aggregation of rankings for multiple constraints*

In systems with multiple constraints (such as the running example) each constraint might yield a different ranking for the same set of uncertainties. Therefore, we construct an overall ranking by aggregating rankings from individual constraints. To this end, we focus only on the uncertainties globally impacting confidentiality violations. Specifically, to normalize each constraint-specific ranking, we sum all rankings and divide by the total, ensuring that all scores range from 0 to 1. This normalization maintains comparability across constraints which exhibit varying levels of severity in violations. We evaluated several aggregation methods and found that each offers unique benefits:

5.3.1. *Simple summation*

This method sums the normalized scores of each constraint-specific ranking, producing a cumulative score. While straightforward in the execution, this approach inflates uncertainties that, in truth, are not critical to any specific violation because only moderately important across multiple constraints.

5.3.2. *Exponential decay*

Applying e^{-r} to the summative rank r of each uncertainty reduces the impact of lower-ranked uncertainties. This approach emphasizes the top-ranked uncertainties of each constraint, thus providing a clearer focus on the most significant factors.

5.3.3. *Top-3 emphasis*

To highlight only the most critical uncertainties, scores are assigned to only the top-three-ranked uncertainties for each constraint, setting all others to zero. Clearly, this approach risks missing lower-ranked yet impactful uncertainties, but as a result, the approach also reduces noise from uncertainties with minimal contributions to constraint violations.

Top-3 Emphasis unifies constraint-specific rankings to identify, across complex models, the uncertainties with the greatest potential for mitigating violations in confidentiality or compliance. While the respective optimal aggregation strategy depends on the structure of the DFD and

Table 4
Aggregated importance of uncertainties for the running example.

Uncertainty	Importance
U1	1.3678
U4	1.0000
U2	0.3678
U3	0.1353

also on the nature of the constraints, our evaluation shows that exponential decay results in the overall best rankings. To illustrate the aggregation, we use the individual importance from the running example (as in Table 3) but aggregated using exponential decay. The results (shown in Table 4) confirm that **U1** has the most impactful importance because it can lead to violations of both constraints.

6. Mitigation strategies

We mitigate confidentiality violations by configuring the software architecture to use a combination of uncertainty scenarios that comply with confidentiality requirements. Our approach aims to find a violation-free model by generating and evaluating new models in which high-ranking uncertainties are iteratively combined and tried out. We iteratively replace uncertainties with their concrete scenarios to produce new model variations. We then check each model for confidentiality violations. If we discover a violation-free model, the mitigation is successful, and the approach terminates. If not, the approach continues, using other uncertainties in the ranking until no further options are available. We want to note, though, that our approach fails if no combination of uncertainty scenarios satisfies all confidentiality requirements. The chosen mitigation strategy determines how many uncertainties we use per model generation. In our previous work (Niehues et al., 2025), we presented the *Depth-First-Search* strategy, the *incremental increase* strategy and *fixed subset* strategies. This paper extends these with the *Fast Start* strategy (Section 6.3) and the *Clustering* strategy (Section 6.5).

6.1. *Depth-first-search strategy*

The simplest strategy involves trying all combinations of uncertainties ranked by appearance in the TFGs, allowing a straightforward analysis without additional overhead. This depth-first-search strategy often works well for small models, as it avoids the time-consuming task of generating training data and ranking the importance of uncertainties. However, this strategy may lead to exponentially increasing runtime if critical uncertainties appear closer to the end of the TFG. In such cases, delays are significant, as more extensive analysis is required to evaluate later uncertainties in the sequence. The DFS strategy serves as a baseline for our evaluation.

6.2. *Incremental increase strategy*

In this strategy, uncertainties are iteratively included in small increments: First, only the scenarios of the top-ranked uncertainty are considered, then the combination of the top two, then the top three, adding one uncertainty per iteration. We generate and analyze a new model variant for each unique scenario combination of these uncertainties. For the running example, we would check all scenario combinations for {**U1**}, then {**U1**,**U4**}, then {**U1**,**U4**,**U2**} and finally {**U1**,**U4**,**U2**,**U3**}. Using the *public* data scenario for **U1** already satisfies all constraints, so we can stop after the first iteration. This strategy is efficient when constraint violations are caused by a few high-ranking uncertainties. However, frequently generating and evaluating models may increase runtime compared to other strategies.

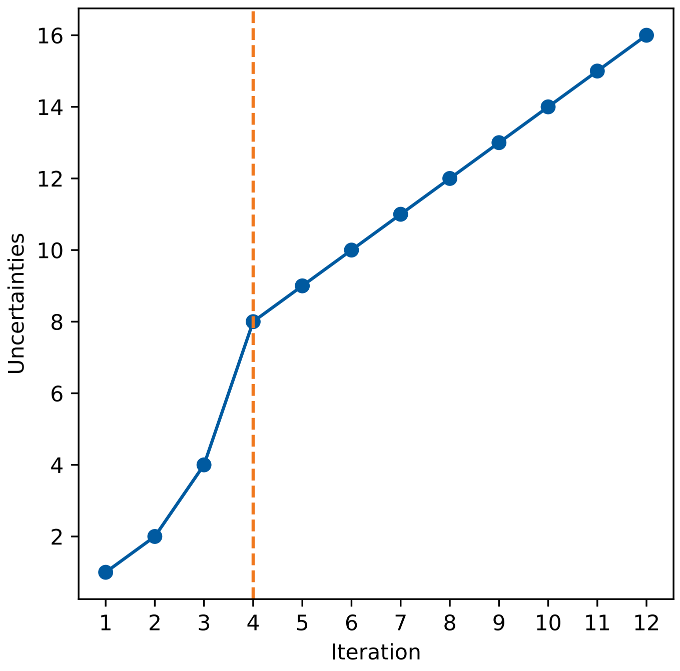


Fig. 3. Combined uncertainties per iteration in the Fast Start approach, with exponential growth up to a threshold, followed by incremental increases.

6.3. *Fast start strategy*

This strategy extends the incremental increase and is inspired by the *TCP Congestion Control* (Blanton et al., 2009). In the first phase, we increase the amount of combined uncertainties exponentially until we reach a threshold. We use half of the total number of uncertainties as the threshold. Afterward, we continue to include uncertainties incrementally. As before, we halt the approach when we find a valid configuration. Fig. 3 shows an example for 16 uncertainties with a threshold of 8.

The main drawback of the *Incremental Increase* strategy is that when numerous uncertainties cause constraint violations, many unnecessary model variants are generated early in the approach. The *Fast Start* strategy mitigates this inefficiency by leveraging the fact that small uncertainty combinations can be evaluated quickly. By initially growing exponentially, we rapidly approach a meaningful threshold before switching to incremental increases. This balances computational efficiency with the need to explore impactful uncertainty combinations while minimizing redundant evaluations.

6.4. *Fixed subset strategies*

With a good ranking, we can assume that constraint-causing uncertainties will likely appear in the top-ranked portion. Therefore, a subset-based strategy can improve efficiency by reducing redundant combination checks. In this strategy, uncertainties are split into varying amounts of batches, for example, into four, three, or two batches. We evaluate each subset by testing all scenario combinations within that batch until we find a violation-free model, allowing the approach to halt early. If we do not find a solution within the current batch, we include the next subset, expanding the pool of uncertainties to test additional combinations. When using two batches for the running example, we would check all scenario combinations for {**U1**,**U4**}, and then {**U1**,**U4**,**U2**,**U3**}. As we can satisfy all constraints within the first batch, we can stop the approach. This strategy performs efficiently when there are many violations causing uncertainties that appear high in the ranking. However, if some of these uncertainties rank lower than half, the runtime increases as more subsets are evaluated.

6.5. *Clustering strategy*

This strategy extends the fixed subset strategy by dynamically determining batch sizes based on the importance of uncertainties using clustering. Specifically, we apply k-means clustering (Kodinariya et al., 2013) on the uncertainty importance scores to form clusters, which are then processed similarly to fixed batches.

To determine the optimal number of clusters (k), we use the Silhouette coefficient, which measures how well the clusters align with uncertainty importance (Kodinariya et al., 2013). We tested values of k ranging from 2 to 8, and our experiments indicated that $k = 8$ provided the best results for our evaluation scenarios.

By dynamically adjusting batch sizes based on model-specific uncertainty importance, this approach combines the advantages of batch-based testing while ensuring better alignment with the given model's structure.

6.6. *Optimizing modified uncertainties*

We defined an additional approach displayed in Algorithm 3 to minimize unnecessary uncertainty modifications that refines the model by merging versions that differ by only a single uncertainty. The algorithm works in combination with the previous strategies and returns a list of relevant scenarios, marks irrelevant uncertainties. It is especially useful for architects who want to retain as much modeled uncertainty as possible. The algorithm identifies and merges models that differ by one uncertainty, reducing unnecessary modifications. If two models contain all possible scenarios of a particular source, the algorithm marks that source as irrelevant for constraint violations. Otherwise, the algorithm merges the models to include the scenarios used for that source. We iterate the approach until no further optimizations are possible.

Algorithm 3 Optimize amount of uncertaintes.

```
 1: function SIMPLIFYMITIGATION(models : List of String, scenarioAmounts : List of Integer)
 2:     newModels ← models
 3:     changeHappened ← true
 4:     while changeHappened do
 5:         changeHappened ← false
 6:         for all pairs of models $(m_i, m_j)$ in newModels do
 7:             if models differ by one uncertainty $u$ then
 8:                 mergedModel ← $m_i \cup m_j$
 9:                 if mergedModel satisfies scenarios then
10:                     $u$ ← irrelevant
11:                 end if
12:                 newModels ← newModels ∪ mergedModel
13:                 newModels ← newModels $\setminus m_i \setminus m_j$
14:                 changeHappened ← true
15:             end if
16:         end for
17:     end while
18:     return newModels
19: end function
```

Through these strategies, the mitigation approach offers flexible ways to reduce runtime and ensure minimal alterations to the model, balancing effectiveness and efficiency based on the specific characteristics of each ranked uncertainty.

7. Evaluation

Our evaluation aims to assess whether the proposed mitigation approach can accurately identify relevant uncertainties and repair confidentiality violations while scaling well with an increasing number of uncertainties. To answer this, we ran exhaustive experiments on three software architecture models. In this section, we present our plan, design, and results. To close out the section, we cover any possible threats to validity.

7.1. *Goal, questions, and metrics*

We use a Goal-Question-Metric (GQM) plan (Basili and Weiss, 1984; Basili, 1992) to structure the evaluation. To enhance validity, we align our plan with related work (Hahner et al., 2023b,a) and use well-known metrics (Kar et al., 2015).

Our **Goal** is to evaluate the quality of our mitigation approach compared to existing approaches (Hahner et al., 2023b,a). We ask the following questions:

Q1 How precisely do the proposed ranker and aggregation strategies identify relevant uncertainties?
Q2 How effective is the automatic mitigation in repairing confidentiality violations?
Q3 How scalable are the refactored mitigation strategies?
Q4 How scalable are the new proposed mitigation strategies?

Question **Q1** investigates the precision of the ranking and aggregation strategies. We evaluate and compare all previously introduced ranking strategies, i.e., Exploratory Factor Analysis (EFA), Principal Component Analysis (PCA), Random Forests (RF), Linear Regression (LR), Logistic Regression (LGR), and Linear Discriminant Analysis (LDA), see Section 2. Furthermore, we consider the three aggregation strategies introduced in Section 5, i.e., simple summation (SUM), exponential decay (EXP), and top-3 emphasis (Top3). To assess the ranking results, we apply the often used *Precision@K* (**M1.1**) metric that measures "the number of relevant items at the top k positions of a ranked list" (Kar et al., 2015). An uncertainty is relevant, i.e., a *true positive (TP)* if it causes a confidentiality violation. Otherwise, it is irrelevant, i.e., a *false positive (FP)*. For instance, Uncertainty **U1** in our running example is always relevant as it affects both confidentiality requirements regardless of the other uncertainties. We calculate $Precision@K = \frac{TP}{TP+FP} \in [0, 1]$ with K being the rank of the last relevant uncertainty. Otherwise, the ranking could ignore relevant uncertainties and negatively impact the recall, which shall be avoided in confidentiality analysis (Hahner et al., 2023b). Put simply, we investigate how many irrelevant uncertainties were ranked among all relevant uncertainties, where lower is better.

Question **Q2** investigates the effectiveness of our automatic mitigation in producing repaired models without confidentiality violations. To answer this question, we compare the number of confidentiality violations before the mitigation (**M2.1**) and after the mitigation (**M2.2**). Trivially, the mitigation shall remove identified confidentiality violations without reintroducing new violations. Thus, lower is better.

Finally, with questions **Q3** and **Q4** we investigate the scalability of the mitigation. We consider the mitigation strategies that determine how uncertainties are selected for the mitigation, i.e., incremental increase (INCREASE), Fast Start (FASTSTART), fixed subsets splitting the relevant uncertainties into two halves (HALF), or four quarters (QUARTER) and clustering. (CLUSTER), see Section 6. We compare these strategies against each other and against Depth-First-Search (BRUTE), representing the baseline of the ABUNAI approach. Here, we want to asses which strategy is expedient and whether we can outperform the state of the art. We measure the runtime (**M3.1**) of the approach for rising numbers of relevant and irrelevant uncertainties to evaluate the scalability.

7.2. *Scenarios*

We selected three diverse software architectures in the form of DFDs as evaluation scenarios. They differ in size (5 to 18 nodes), interconnectedness, and the number of uncertainties (7 to 21), representing a range

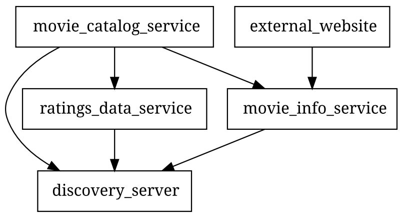


Fig. 4. Spring boot architecture.

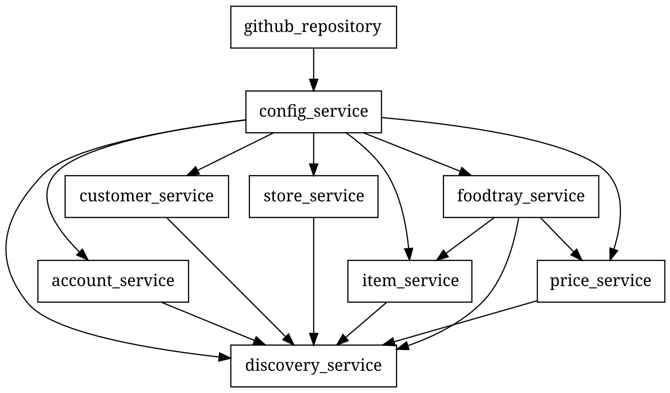


Fig. 5. Tap and eat architecture.

of real-world architectures. These systems also span varied technological and functional domains, including open-source microservices and secure online banking. This heterogeneity demonstrates the applicability of our approach to a broad spectrum of architectures with similar characteristics.

Spring Boot. Kothagal (2024) created this open-source microservice architecture, and Schneider et al. (2023) derived its DFD representation. The model is shown in Fig. 4 and contains 5 nodes representing 1 user interface, 3 internal services and 1 data storage. The nodes are connected by 6 edges. The model contains 7 uncertainties, and 3 out of those cause confidentiality violations. For this model we check the following constraints (Niehues et al., 2024):

$$DataLabel(entrypoint) \implies DataLabel(encrypted\_connection)$$

$$NodeLabel(internal) \implies DataLabel(encrypted\_connection)$$

$$NodeLabel(local\_logging) \implies NodeLabel(internal)$$

Tap and Eat. Ferrater (2024) created this open-source architecture, and Schneider et al. (2023) derived its DFD. The model is shown in Fig. 5 and displays a microservice architecture containing 9 nodes representing 8 internal services and 1 data storage. The nodes are connected by 16 edges. The model contains 21 uncertainties, and 4 out of those cause confidentiality violations. For this model we check the following constraints (Niehues et al., 2024):

$$NodeLabel(internal) \implies DataLabel(auth\_request)$$

$$NodeLabel(login\_attempts\_reg) \implies NodeLabel(auth\_server)$$

$$DataLabel(entrypoint) \implies DataLabel(encrypted\_connection)$$

$$NodeLabel(local\_logging) \implies NodeLabel(internal)$$

Online Banking. The last model is our own creation and displays an online banking architecture. The model is shown in Fig. 6 and consists of 18 nodes representing 2 external actors, 3 user interfaces, 10 internal services and 3 data storages. The nodes are connected by 20 edges. The model contains 9 uncertainties, and 3 out of those cause confidentiality violations. For this model we check the following constraints:

$$NodeLabel(nonEU) \implies \neg DataLabel(Personal)$$

$$NodeLabel(Processable) \implies \neg DataLabel(Encrypted)$$

$$NodeLabel(Develop) \implies \neg DataLabel(Personal)$$

7.3. Design

To evaluate the precision of our ranking, we created reference rankings. To that end, we manually examined the evaluation scenarios and identified the relevant uncertainties, which make our true positives. Each author, independently of one another, assessed which uncertainties were responsible for the observed confidentiality violations. Next, all authors compared assessments. In all cases, the authors arrived at the same conclusions because the violating uncertainties will be self-evident to any researcher with expertise in software architecture and confidentiality. Next, on the basis of the reference rankings, our automatic mitigation was executed with different combinations of uncertainty ranker and aggregation strategies and calculated the Precision@K score.

To evaluate the effectiveness of our automatic mitigation approach, we employed the ABUNAI framework (Hahner, 2025). The analysis takes a DFDs and a list of constraints as input and automatically checks for violations and their locations. We use the output of this analysis to measure the number of confidentiality violations detected both before and after applying our mitigation.

To evaluate the scalability of our approach, we manually scaled the evaluation scenarios. We iteratively duplicated subgraphs to scale the Spring Boot and Online Banking model up, which increased the nodes and uncertainty counts. Conversely, to scale the Tap and Eat model down, we removed subgraphs step by step. We measured the mitigation runtime in milliseconds on an Apple M3 Pro-CPU with 18 Gigabytes of RAM for every evaluation scenario and scaling step.

7.4. Results

7.4.1. Ranking of uncertainties

We evaluated every uncertainty ranker on every aggregation strategy. In this way, we obtained a total of 54 Precision@K scores, or 18 for the Spring Boot model (see Fig. 7a), 18 for the Tap and Eat model (see Fig. 7b), and 18 for the Online Banking model (see Fig. 7c). In the figures, each bar represents a Precision@K value ranging from 0 to 1, where 1 indicates a perfect score.

Across all evaluation scenarios, supervised rankers consistently outperformed unsupervised rankers. Specifically, Logistic Regression and Random Forest were the top-performing rankers; moreover, the Exponential and Top3 aggregation strategies, when combined with Logistic Regression, consistently produce perfect rankings across all models. In contrast, unsupervised rankers, such as EFA and PCA, underperformed, especially in models with high variance among uncertainties.

All in all, our combination of logistic regression with exponential decay achieved a perfect ranking of uncertainties across all evaluation scenarios.

7.4.2. Effectiveness of our mitigation

To evaluate the effectiveness of our automatic mitigation approach, we employed ABUNAI on three evaluation scenarios and then measured the number of confidentiality violations detected both before and after applying our mitigation.

Table 5 summarizes our results. Prior to mitigation, the Spring Boot model exhibited 3 confidentiality violations, the Tap and Eat model 9 violations, and the Online Banking Model 4 violations. After applying our mitigation approach, all models demonstrated zero confidentiality violations. Thus, our results confirm the effectiveness of our approach in mitigating confidentiality violations.

7.4.3. Scalability of our mitigation

To provide a broader set of data points for evaluating scalability, we modified the models by duplicating or removing subgraphs, thereby

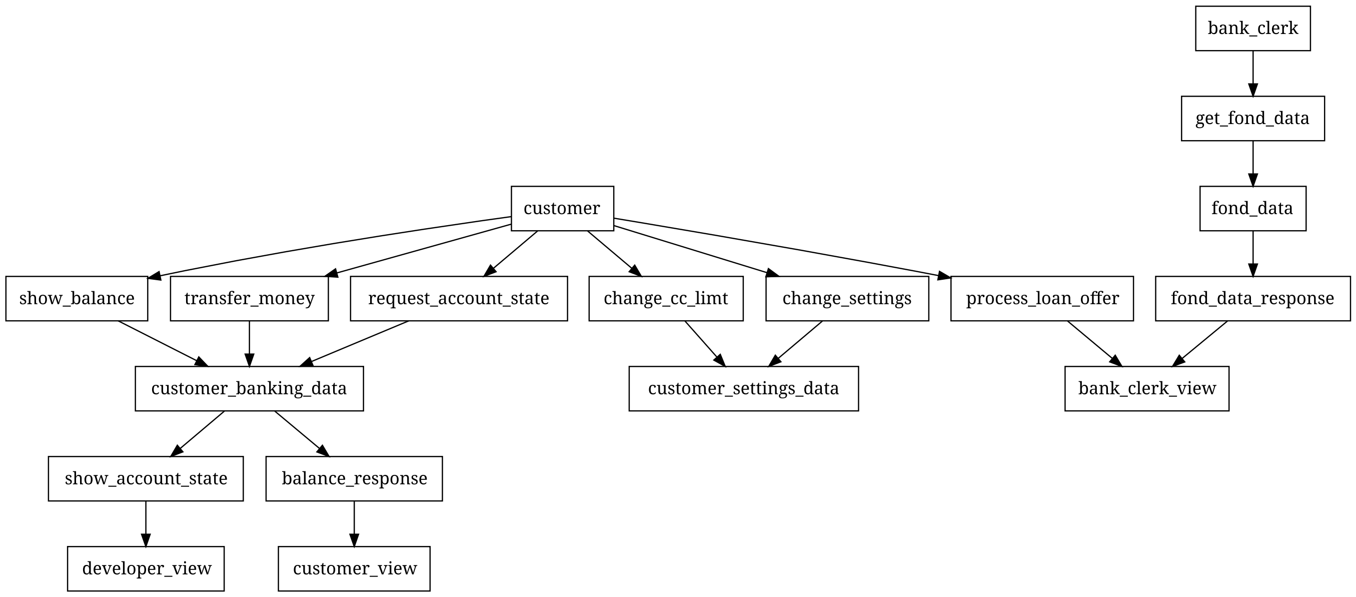


Fig. 6. Online banking architecture.

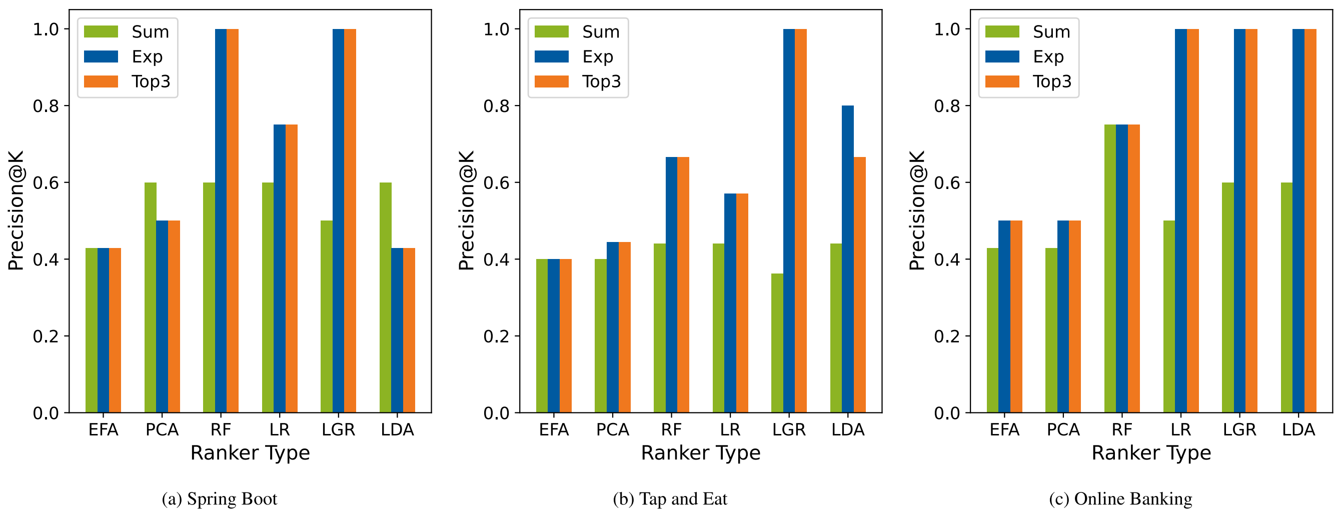


Fig. 7. Precision@K for different uncertainty rankers and aggregation strategies.

Table 5
Confidentiality violations identified before and after mitigation.

	Before	After
Spring Boot Model	3	0
Tap and Eat Model	9	0
Online Banking Model	4	0

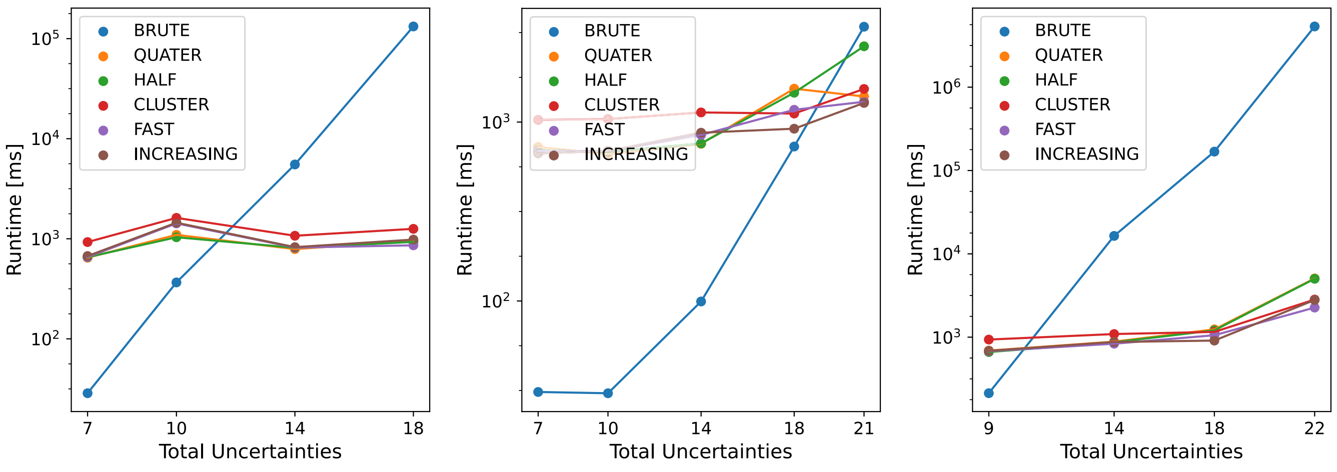
yielding models with uncertainty counts ranging from 7 to 22. To preserve model behavior and structure, we maintained a proportional distribution of relevant to total uncertainties. For each uncertainty level, we executed the mitigation approach multiple times and then recorded the average runtime in milliseconds on a logarithmic scale. Fig. 8 shows our results both for the refactored strategies of Niehues et al. (2025) and also for our two new strategies.

The baseline depth-first search (BRUTE) exhibits exponential growth in runtime as uncertainties increase in number. In the Online Banking model, to resolve the 22 uncertainties, the baseline takes 5356 seconds — that is, 1 h 30 minutes — and so represents the worst case, while our new Fast Start strategy, by contrast, resolves in just 2 seconds. Overall, our mitigation strategies consistently maintain runtimes near 1 second, even for the largest models, and no strategy ever exceeds 10 seconds runtime.

However, our strategies do introduce some overhead by using machine learning for the uncertainty ranking. Consequently, for models with low uncertainty (e.g., fewer than 14 uncertainties), the runtime of the baseline may outperform the runtimes of our strategies.

7.4.4. *Statistical analysis*

Based on the scalability results shown in Section 7.4.3, we propose the following hypothesis:



(a) Spring Boot (b) Tap and Eat (c) Online Banking

Fig. 8. Runtime scalability for different mitigation strategies.

> **Hypothesis:** Our mitigation strategies outperform the depth-first search baseline when there are at least 14 uncertainties.

To test this hypothesis, we conducted the Friedman test to assess whether there are statistically significant differences among strategies. The test yielded a p-value of 0.025 and a Kendall's W of 0.721. Those results indicate a significant difference and a strong effect size, and so we conclude that the null hypothesis is rejected.

Next, we proceeded with a Nemenyi post hoc test. We treated strategies as groups, we treated runtime as the target metric, and we treated evaluation scenarios across different amounts of uncertainty as blocks. Table 6 summarizes all resulting p-values. We set the significance level to $\alpha = 0.05$ and found that the Fast Start strategy significantly outperforms the depth-first search baseline ($p = .039$). However, under the $\alpha = 0.05$ threshold, no other pairwise differences were statistically significant, neither against the baseline nor between any of the other pairs.

All in all, on smaller models, our mitigation approaches may underperform the baseline because of the overhead introduced by the machine-learning techniques. Yet, as the number of uncertainties reaches 14, our mitigation approach outperforms the baseline, as demonstrated on a logarithmic scale in Fig. 9. Our approaches achieve a median elevenfold runtime reduction; moreover, using our new Fast Start strategy, our approaches achieve a maximum of a 2298-fold runtime reduction on the Online Banking model with 22 uncertainties.

7.5. Threats to validity

We structure the discussion of threats to validity based on the guidelines by Runeson and Höst (2009).

Table 6
Nemenyi Post-hoc p-values.

	Brute	Quarter	Half	Cluster	Fast	Increasing
Brute	1.000	0.985	0.596	0.995	**0.039**	0.206
Quarter	0.985	1.000	0.937	1.000	0.206	0.596
Half	0.596	0.937	1.000	0.894	0.765	0.985
Cluster	0.995	1.000	0.894	1.000	0.154	0.507
Fast	**0.039**	0.206	0.765	0.154	1.000	0.985
Increasing	0.206	0.596	0.985	0.507	0.985	1.000

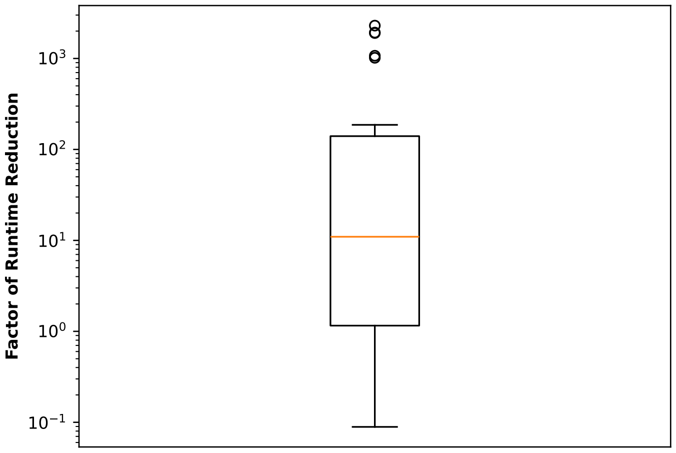


Fig. 9. Factor of runtime reduction for models with at least 14 uncertainties.

Internal validity. One threat is the accuracy of the model-based confidentiality analysis (Hahner et al., 2023a) used to assess confidentiality violations in the repaired models. Wrong results would negatively affect the correctness of our results. However, as this analysis has already been comprehensively evaluated, we consider this threat to be negligible. Another threat is the subjectivity of our manual identification of relevant uncertainties in the evaluation scenarios. To mitigate this, each author independently assessed which parts were responsible for the confidentiality violations, and then all authors compared results. We consistently reached the same conclusions, since the problematic elements were largely self-evident to anyone familiar with confidentiality requirements. Note, too, that this manual assessment is only one part of our evaluation of the quality of the ranking; the assessment does not reflect on the ability of our approach to mitigate confidentiality violations. Therefore, any possible impact of this threat is even further limited.

External validity. The choice of cases and evaluation scenarios can limit the generalizability of our results. To address this, we choose models from diverse backgrounds, with the two from Schneider et al. (2023) being derived from real-world open-source projects.

Construct validity. We applied a GQM plan with well-known metrics such as Precision@K. Last, we involved multiple researchers in the process to enhance the *reliability* of our results and minimize the bias in annotating our models. Every step in our process is deterministic, given the same input model. Therefore, the only variance in our runtime measurements stems from differing CPU loads. To reduce this variability, we averaged runtimes from twelve runs per measurement. Last, we provide supplemental material as data set (Niehues et al., 2025) to address the availability of evolution artifacts in software architecture research (Konersmann et al., 2022). This includes all code artifacts, results, and instructions on reproducing them.

8. Discussion

8.1. Confidentiality for trustworthier software

Here we revisit four sets of our results, in order to draw out exactly how these help our extended approach to contribute to trustworthier software.

First, one set of results on ranking uncertainties which cause violations (detailed in Section 7.4.1) shows that supervised machine learning techniques produce better rankings than unsupervised techniques. We infer that supervised rankers align so well with this prediction task that they are capable of capturing more effectively the relevant uncertainties. On the other hand, unsupervised rankers appear to rely too heavily on shared variance, which can introduce noise when unrelated variables correlate with the target constraints. Therefore, supervised techniques are better at identifying relevant uncertainties than unsupervised techniques.

Second, another set of results on ranking uncertainties (again, detailed in Section 7.4.1) shows that across all models, the Exponential aggregation performs best, while the Sum method performs worst. The Exponential method possesses the ability to reduce noise by weighting uncertainties exponentially. Conversely, the Sum method underperforms, likely because it evenly distributes importance and thus introduces random noise and reduces precision. Therefore, exponential decay is better suited to aggregate individual importance than just summing up importance, constraint by constraint.

Third, one set of results on scalability (detailed in Section 7.4.3) shows that our mitigation strategies, when compared against the depth-first search baseline ABUNAI (Hahner, 2025), are much faster, and in the case of the Fast Start strategy, even significantly faster. We attribute this considerable gain in efficiency to the effective ranking of uncertainties. Effective rankings allow our strategies to consider only those scenario combinations which are necessary to mitigate confidentiality violations. In contrast, the baseline ABUNAI must first inspect many unnecessary combinations before it gets to the root of the problem and mitigates the violations. Therefore, trying exclusively combinations of the relevant uncertainties is faster than trying combinations based on appearance in the model.

Lastly, another set of results on scalability (again, detailed in Section 7.4.3) shows that with our extension, we can reduce the runtime compared to the baseline by a factor of up to 2298 times. In Niehues et al. (2025), we had achieved merely a 60-fold reduction on the same model with the same amount of uncertainties. We explain this extraordinary improvement in two ways. First, we meticulously improved our prototype to increase code quality, to remove redundant operations, and to make it more efficient. Second, the newly added Fast Start strategy, which is based on the TCP Congestion Control, is not only the fastest strategy, but also significantly faster than the baseline with a p value below 0.05 and Kendall's W of 0.721 (i.e., a strong impact). Therefore, by polishing the prototype and also by developing more advanced mitigation strategies, we transformed our previous work through dramatic improvement in runtime.

In sum, this extension of our previous work (Niehues et al., 2025) is a significant contribution both specifically to the mitigation of confidentiality violations and generally to the trustworthiness of software architectures.

8.2. Discrete quality attributes for trustworthier software

Trustworthiness is "The degree to which a system deserves to be trusted based on the qualities of security, safety, reliability, resilience, and privacy" (ISO/IEC, 2011). Clearly, trustworthiness will be strongly influenced by confidentiality, because this attribute is an integral part of security. However, confidentiality is just one among a number of additional relevant quality attributes. For example, the attributes integrity, authenticity, and privacy will all influence trustworthiness too, and interestingly, the ways that they influence trustworthiness may prove to be similar or even identical to how confidentiality influences trustworthiness. The reason is that attributes like integrity, authenticity, and privacy are binary in nature and thus lack gradients for incremental improvement. In contrast to a continuous attribute like reliability, binary attributes like these are harder to optimize for efficiently because they are discrete (Aspvall and Stone, 1980; Lenstra, 1983). Therefore the discreteness of an attribute actually presents a major challenge for violations mitigation generally, and thus impinges upon the trustworthiness of the software.

Our approach promises to deliver effective application also to other discrete quality attributes because our mitigation of confidentiality violations is agnostic to concrete components, to their behavior, and to the rules imposed on them. As long as (1) a model and (2) the uncertainties of that model and (3) the constraints in the underlying analysis are all well-defined, then our approach can use those outputs to rank uncertainties and combine the uncertainty scenarios for our violations mitigation. That, of course, is a noteworthy advance on ensuring generally – beyond just confidentiality – greater trustworthiness in software systems.

Our approach, again, should apply to any *discrete* quality attribute *when these two prerequisites are met*, namely, that the quality attribute be constrainable in a formal way and also that there be an analysis which identifies violations of the constraints under uncertainty. Such an analysis can be either sourced from related work or built as an adaptation of our confidentiality analysis here. If the analysis is adapted from this work, the DFD metamodel only needs to be adjusted to incorporate the additional properties and behaviors of whichever quality attribute is under analysis. For example, if the quality attribute for analysis is integrity, then the DFD will need to be made capable of saving hashes for data sent between nodes. This way, the analysis of integrity can now verify whether the data was changed between the source node and the sink node. Such a step in analysis is all that should be needed to mitigate violations of any discrete quality attribute just as effectively as we have demonstrated that our approach mitigates violations in confidentiality constraints.

We look forward to future work in this direction. We first plan to extend our approach to the attributes of authenticity, integrity and privacy. Advances here should help achieving *generally* trustworthier software.

9. Related work

In this paper, we consider uncertainty within the software architecture to enable the automated repair of confidentiality violations. We identify three areas of work related to this paper. First, work from the field of software architecture research that deals with uncertainty, which is often related to self-adaptive systems (Hezavehi et al., 2021; Weyns, 2020; Weyns et al., 2023). Second, approaches to confidentiality analysis at design time that enable the early identification of confidentiality violations (Seifermann et al., 2022; Boltz et al., 2023, 2020, 2024), also while considering uncertainty (Hahner et al., 2023b,a). Third, using machine learning to mitigate security issues (Ahsan et al., 2022; Kronjee et al., 2018; Steenhoek et al., 2024). We summarize these research directions in the following.

Architectural analysis and mitigation under uncertainty. Numerous papers aim to understand better the representation and impact of uncertainty on architectural models. Troya et al. (2021) performed a survey to investigate the representation of uncertainty in models and found that modeling uncertainty as variation models and scenarios is common, especially in design space exploration. Andersson et al. (2009) support this view by presenting modeling dimensions of self-adaptive systems, as does the recently proposed OMG *PSUM* standard (Group, 2023). Researchers have proposed numerous classifications (Perez-Palacin and Mirandola, 2014; Mahdavi-Hezavehi et al., 2017; Ramirez et al., 2012) to investigate the nature of uncertainty, including some focused on security-related fields like access control (Bures et al., 2020) and confidentiality (Hahner et al., 2023). Comprehensive frameworks have been researched to investigate uncertainty in software systems, e.g., *RELAX* (Whittle et al., 2009), Rainbow (Garlan et al., 2004), PerOpteryx (Koziolek et al., 2011), ArcheOpterix (Aleti et al., 2009), GuideArch (Esfahani et al., 2013), and DeTUM (Famelis and Chechik, 2019). Here, Hezavehi et al. (2021) and Sobhy et al. (2021) recently conducted surveys and found that addressing uncertainty in design time is expedient and mitigation should be systematically considered. However, most existing approaches focus only on analysis rather than mitigation and do not explicitly support confidentiality. This can severely limit their applicability (Walter et al., 2022b; Hahner et al., 2023a) to identify and repair confidentiality violations under uncertainty. Researchers have highlighted the challenge of creating comprehensive end-to-end approaches (Weyns et al., 2023). Here, especially considering multiple uncertainty sources and their interactions is considered to be challenging (Cámara et al., 2022; Camara et al., 2024). The work presented in this paper addresses this gap by considering both the analysis and the mitigation of uncertainty at design time, which is tailored to confidentiality violations.

Architecture-based confidentiality analysis. Numerous approaches have been proposed to identify confidentiality violations using the architectural abstraction, e.g., data flow-based confidentiality analysis (Seifermann et al., 2022, 2021; Boltz et al., 2023), or architecture-based access control analysis (Walter et al., 2022b, 2023). Furthermore, broader approaches to model-based security analysis, e.g. UMLsec (Jürjens, 2002), or SecDFD (Tuma et al., 2019; Peldszus et al., 2019). Despite focussing on security, these approaches lack support for confidentiality or automated model repair. More recently, uncertainty-aware confidentiality analysis has been proposed, e.g., by extending the PerOpteryx framework (Walter et al., 2022a), by tracing uncertainty in data flow diagrams (Hahner et al., 2023a), or by combining data flow analysis with fuzzy inference to represent uncertainty (Boltz et al., 2022). Hahner et al. (2023b) proposed an uncertainty impact analysis to predict uncertainty's potential impact on software architectures' confidentiality. However, these approaches only focus on analyzing software architectures without considering mitigating confidentiality violations. The work presented in this paper addresses this by combining analysis and mitigation into one comprehensive approach.

Machine learning addressing security. More recently, machine learning has been used more thoroughly to address security issues such as confidentiality violations. Ahsan et al. (2022) give an overview of common security threats and which machine learning techniques can be used to mitigate them, e.g., deep learning or reinforcement learning. One example is the work of Kronjee et al. (2018), who extract security-related features from source code to use in models like decision trees and random forests. Another recently proposed approach by Steenhoek et al. (2024) proposed combining classical program analysis, such as data flow analysis, with deep learning to increase the accuracy while reducing the runtime. Although applying machine learning seems to be expedient, these approaches lack the required explainability (Hahner et al., 2024; Bersani et al., 2023) of issues to software engineers and the abstraction of the software architecture. Choosing the right abstraction and representation to investigate confidentiality greatly impacts the understandability of security experts (Schneider et al., 2024). Furthermore, only considering source code in the analysis limits the applicability at design time, which is required for early mitigation to minimize costs (Boehm and Basili, 2001). We address this by incorporating machine learning into data flow analysis at design time and using this abstraction to enhance the explainability of identified and repaired confidentiality violations.

10. Conclusion

In this paper, we contribute a novel, scalable solution for mitigating confidentiality violations in software architectures under uncertainty, underlining the benefits of combining architecture-based analysis with machine learning techniques to support secure software development.

The approach proposed here extends our previous machine-learning-based approach for mitigating confidentiality violations in software architectures (Niehues et al., 2025). Specifically, our extension here advances beyond the original because we introduce two major new strategies. First, we take inspiration from TCP Congestion Control and so are able to significantly accelerate the identification of critical uncertainties. Second, we now make allowance for the dynamics of uncertainty sources and so develop a clustering-based technique for adjusting batch sizes dynamically to the structure of the input model. Taken together, these changes over our previous approach deepen our understanding about the nature of uncertainty and so, too, about those techniques optimally suited to mitigating the violations caused by uncertainties.

The evaluation in this work is much more rigorous than in Niehues et al. (2025) because we increased the amount of data points and also conducted a thorough statistical analysis (including the Friedman test and the Nemenyi post hoc test). Our previous work (Niehues et al., 2025) achieved a mere 60 times mitigation runtime reduction over the state-of-the-art. This extension of that work, on the other hand, has a median reduction that is elevenfold but achieves an impressive maximum of 2298 times runtime reduction *on the same model with the same level of uncertainty*. This big advance over the state-of-the-art was made possible by our two new strategies and as well by our careful polishing of the prototype. Moreover, we obtain significantly faster mitigation times compared to the state-of-the art baseline, when at least 14 uncertainties are present. We found a significant difference (Friedman test $p = .025$) between our strategies and the baseline and as well, a strong practical impact (Kendall's $W = 0.721$). Further, our new TCP-inspired strategy proved to be significantly faster than the baseline (Nemenyi post hoc test $p = .039$).

Our novel automatic mitigation succeeds in advancing existing analysis frameworks beyond just the identification of confidentiality violations so that now we can truly assist software architects in efficiently addressing confidentiality challenges at design time. Our approach automates uncertainty prioritization and resolution, thereby enhancing the scalability of confidentiality maintenance in complex systems to ensure that data protection measures are more manageable and reliable. That is clearly a step forward in trustworthier software generally. Additionally, our extended approach opens the door to similar approaches to violations mitigation for other quality attributes which are discrete like confidentiality. A promising direction for future work is to extend these ideas to attributes such as authenticity, integrity, and privacy.

Future work might also attempt to automatically generate scenarios using machine learning trained on real-world software architectures. That would help to further reduce the manual effort of software architects. Additionally, we plan to explore combining SAT solvers with fine-grained cost estimation, in order to identify a configuration that satisfies all confidentiality requirements while minimizing the cost of the mitigation.

CRediT authorship contribution statement

Nils Niehues: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Data curation, Conceptualization; **Sebastian Hahner:** Writing – original draft, Supervision, Software, Methodology; **Robert Heinrich:** Writing – review & editing, Writing – original draft, Supervision.

Data availability

I have provided supplementary material hosted on zenodo that includes artefacts and results.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgement

This work was supported by the topic Engineering Secure Systems of the Helmholtz Association (HGF) and by KASTEL Security Research Labs, the BMBF (German Federal Ministry of Education and Research) grant number 16KISA086 (ANYMOS), and the NextGenerationEU project by the European Union (EU). This paper has been edited by our textician, Daniel Shea. We would like to thank Benjamin Arp.

References

Abdi, H., Williams, L.J., 2010. Principal component analysis. Wiley Interdiscip. Rev. Comput. Stat. 2 (4), 433–459.

Ahsan, M., Nygard, K.E., Gomes, R., Chowdhury, M.M., Rifat, N., Connolly, J.F., 2022. Cybersecurity threats and their mitigation approaches using machine learning-a review. J. Cybersecur. Priv. 2 (3), 527–555.

Aleti, A., Bjornander, S., Grunske, L., Meedeniya, I., et al., 2009. Archeopterix: an extendable tool for architecture optimization of AADL models. In: 2009 ICSE Workshop on Model-Based Methodologies for Pervasive and Embedded Software, pp. 61–71. https://doi.org/10.1109/MOMPES.2009.5069138

Andersson, J., de Lemos, R., Malek, S., Weyns, D., et al., 2009. Modeling dimensions of self-adaptive software systems. In: Cheng, B. H.C., de Lemos, R., Giese, H., Inverardi, P., Magee, J. (Eds.), Software Engineering for Self-Adaptive Systems. Springer, pp. 27–47. https://doi.org/10.1007/978-3-642-02161-9_2

Aspvall, B., Stone, R.E., 1980. Khachiyan'S linear programming algorithm. J. Algo. 1 (1), 1–13.

Balakrishnama, S., Ganapathiraju, A., 1998. Linear discriminant analysis-a brief tutorial. Institute for Signal and information Processing 18 (1998), 1–8.

Basili, V.R., 1992. Software Modeling and Measurement: The Goal/Question/Metric Paradigm. Technical Report. USA.

Basili, V.R., Weiss, D.M., 1984. A methodology for collecting valid software engineering data. IEEE Trans. Software Eng. SE-10 (6), 728–738. https://doi.org/10.1109/TSE.1984.5010301

Bersani, M.M., Camilli, M., Lestingi, L., Mirandola, R., Rossi, M., Scandurra, P., et al., 2023. A conceptual framework for explainability requirements in software-intensive systems. In: 2023 IEEE 31St International Requirements Engineering Conference Workshops (REW). IEEE, pp. 309–315. https://doi.org/10.1109/REW57809.2023.00059

Blanton, E., Paxson, D.V., Allman, M., 2009. TCP Congestion Control. RFC 5681. https://doi.org/10.17487/RFC5681

Boehm, B., Basili, V.R., 2001. Software defect reduction top 10 list 34 (1), 135–137. Conference Name: Computer. https://doi.org/10.1109/2.962984

Boltz, N., Hahner, S., Gerking, C., Heinrich, R., 2023. An extensible framework for architecture-based data flow analysis for information security. In: European Conference on Software Architecture. Springer, pp. 342–358.

Boltz, N., Hahner, S., Walter, M., Seifermann, S., Heinrich, R., Bures, T., Hnetynka, P., et al., 2022. Handling environmental uncertainty in design time access control analysis. In: 2022 48th Euromicro Conference on Software Engineering and Advanced Applications (SEAA). IEEE, pp. 382–389. https://doi.org/10.1109/SEAA56994.2022.00067

Boltz, N., Schmid, L., Taghavi, B., Gerking, C., Heinrich, R., et al., 2024. Modeling and analyzing zero trust architectures regarding performance and security. In: Galster, M., Scandurra, P., Mikkonen, T., Oliveira Antonino, P., Nakagawa, E.Y., Navarro, E. (Eds.), Software Architecture. Springer Nature Switzerland, pp. 253–269. https://doi.org/10.1007/978-3-031-70797-1_17

Boltz, N., Walter, M., Heinrich, R., et al., 2020. Context-based confidentiality analysis for industrial IoT. In: 2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), pp. 589–596. https://doi.org/10.1109/SEAA51224.2020.00096

Breiman, L., 2001. Random forests. Mach. Learn. 45, 5–32.

Bures, T., Hnetynka, P., Heinrich, R., Seifermann, S., Walter, M., et al., 2020. Capturing dynamicity and uncertainty in security and trust via situational patterns. In: Margaria, T., Steffen, B. (Eds.), Leveraging Applications of Formal Methods, Verification and Validation: Engineering Principles. Springer International Publishing, pp. 295–310. https://doi.org/10.1007/978-3-030-61470-6_18

Camara, J., Hahner, S., Perez-Palacin, D., Vallecillo, A., Acosta, M., Bencomo, N., Calinescu, R., Gerasimou, S., et al., 2024. Uncertainty flow diagrams: towards a systematic representation of uncertainty propagation and interaction in adaptive systems. In: Proceedings of the 19th International Symposium on Software Engineering for Adaptive and Self-Managing Systems. Association for Computing Machinery, pp. 37–43. https://doi.org/10.1145/3643915.3644084

Cámara, J., Troya, J., Vallecillo, A., Bencomo, N., Calinescu, R., Cheng, B. H.C., Garlan, D., Schmerl, B., et al., 2022. The uncertainty interaction problem in self-adaptive systems 21 (4), 1277–1294. https://doi.org/10.1007/s10270-022-01037-6

DeMarco, T., 1979. Structured analysis and system specification. Pioneers and Their Contributions to Software Engineering. Prentice-Hall. 255–288.

Esfahani, N., Malek, S., Razavi, K., et al., 2013. Guidearch: guiding the exploration of architectural solution space under uncertainty. In: 2013 35th International Conference on Software Engineering (ICSE), pp. 43–52. ISSN: 1558-1225. https://doi.org/10.1109/ICSE.2013.6606550

Famelis, M., Chechik, M., 2019. Managing design-time uncertainty 18 (2), 1249–1284. https://doi.org/10.1007/s10270-017-0594-9

Ferrater, J., 2024. Tap-and-eat-microservices. Accessed: 2024-11-16. https://github.com/jferrater/Tap-And-Eat-MicroServices.

Garlan, D., Cheng, S.-W., Huang, A.-C., Schmerl, B.R., Steenkiste, P., 2004. Rainbow: architecture-based self-adaptation with reusable infrastructure 37 (10), 46–54. https://doi.org/10.1109/MC.2004.175

Glinz, M., 2006. Requirements engineering i. Nicht funktionale Anforderungen. Universität Zürich, Institut für Informatik, Zürich.

Group, O.M., 2023. Precise semantics for uncertainty modeling (PSUM), version 1.0 beta 2. https://www.omg.org/spec/PSUM/1.0/Beta2/PDF.

Hahner, S., 2021. Dealing with uncertainty in architectural confidentiality analysis. In: Proceedings of the Software Engineering 2021 Satellite Events. Gesellschaft für Informatik, pp. 1–6.

Hahner, S., 2025. Architecture-Based and Uncertainty-Aware Confidentiality Analysis. Ph.D. thesis. Karlsruher Institut für Technologie (KIT). https://doi.org/10.5445/IR/1000178700

Hahner, S., Bitschi, T., Walter, M., Bureš, T., Hnětynka, P., Heinrich, R., et al., 2023a. Model-based confidentiality analysis under uncertainty. In: 2023 IEEE 20th International Conference on Software Architecture Companion (ICSA-C). IEEE, pp. 256–263. https://doi.org/10.1109/ICSA-C57050.2023.00062

Hahner, S., Heinrich, R., Reussner, R., et al., 2023b. Architecture-based uncertainty impact analysis to ensure confidentiality. In: 2023 IEEE/ACM 18th Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS). IEEE/ACM, pp. 126–132. ISSN: 2157–2321. https://doi.org/10.1109/SEAMS59076.2023.00026

Hahner, S., Niehues, N., Boltz, N., Fuksa, M., Heinrich, R., 2024. Arc^{3}n: a collaborative uncertainty catalog to address the awareness problem of model-based confidentiality analysis. In: ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems (MODELS Companion '24). ACM. https://doi.org/10.1145/3652620.3688556

Hahner, S., Seifermann, S., Heinrich, R., Reussner, R., et al., 2023. A classification of software-architectural uncertainty regarding confidentiality. In: E-Business and Telecommunications. Springer Nature Switzerland, pp. 139–160. https://doi.org/10.1007/978-3-031-36840-0_8

Hezavehi, S.M., Weyns, D., Avgeriou, P., Calinescu, R., Mirandola, R., Perez-Palacin, D., 2021. Uncertainty in self-adaptive systems: a research community perspective 15 (4). https://doi.org/10.1145/3487921

Hooper, D., 2012. Exploratory factor analysis.

Imp, 2024. Feature importances with a forest of trees. https://scikit-learn.org/stable/auto_examples/ensemble/plot_forest_importances.html. Accessed: 2024-08-12.

ISO/IEC, M., 2011. Systems and software engineering–systems and software quality requirements and evaluation (SQuaRE)–system and software quality models.

ISO/IEC, 2018. ISO/IEC 27000:2018(e) information technology - security techniques - information security management systems - overview and vocabulary.

Jürjens, J., 2002. UMLsec: Extending UML for secure systems development. In: Jézéquel, J.-M., Hussmann, H., Cook, S. (Eds.), UML 2002 - the Unified Modeling Language. Springer Berlin Heidelberg. Vol. 2460, pp. 412–425. Series Title: Lecture Notes in Computer Science. https://doi.org/10.1007/3-540-45800-X_32

Kar, P., Narasimhan, H., Jain, P., 2015. Surrogate functions for maximizing precision at the top. In: Bach, F., Blei, D. (Eds.), Proceedings of the 32Nd International Conference on Machine Learning. PMLR, Lille, France, pp. 189–198. https://proceedings.mlr.press/v37/kar15.html.

Kodinariya, T.M., Makwana, P.R., et al., 2013. Review on determining number of cluster in k-means clustering. Int. J. 1 (6), 90–95.

Konersmann, M., Kaplan, A., Kühn, T., Heinrich, R., Koziolek, A., Reussner, R., Jürjens, J., al Doori, M., Boltz, N., Ehl, M., Fuchs, D., Groser, K., Hahner, S., Keim, J., Lohr, M., Sağlam, T., Schulz, S., Töberg, J.-P., et al., 2022. Evaluation methods and replicability of software architecture research objects. In: 2022 IEEE 19th International Conference on Software Architecture (ICSA). IEEE, pp. 157–168. https://doi.org/10.1109/ICSA53651.2022.00023

Kothagal, K., 2024. Spring boot microservices workshop. Accessed: 2024-11-16. https://github.com/koushikkothagal/spring-boot-microservices-workshop.

Koziolek, A., Koziolek, H., Reussner, R., et al., 2011. Peropteryx: automated application of tactics in multi-objective software architecture optimization. In: Proceedings of the Joint ACM SIGSOFT Conference – QoSA and ACM SIGSOFT Symposium – ISARCS on Quality of Software Architectures – QoSA and Architecting Critical Systems – ISARCS.

Association for Computing Machinery, pp. 33–42. https://doi.org/10.1145/2000259.2000267

Koziolek, A., 2011. Automated improvement of software architecture models for performance and other quality attributes. https://doi.org/10.5445/IR/1000024955

Kronjee, J., Hommersom, A., Vranken, H., 2018. Discovering software vulnerabilities using data-flow analysis and machine learning. In: Proceedings of the 13th International Conference on Availability, Reliability and Security. Association for Computing Machinery, pp. 1–10. https://doi.org/10.1145/3230833.3230856

Larsen, P.G., Plat, N., Toetenel, H., 1994. A formal semantics of data flow diagrams. Formal Asp. Comput. 6, 586–606.

Lehman, M., Fernáandez-Ramil, J.C., 2006. Software evolution. Softw. Evol. Feedback: Theory pract. , 7.

Lenstra, Jr, H.W., 1983. Integer programming with a fixed number of variables. Math. Oper. Res. 8 (4), 538–548.

Linardatos, P., Papastefanopoulos, V., Kotsiantis, S., 2020. Explainable ai: a review of machine learning interpretability methods. Entropy 23 (1), 18.

Mahdavi-Hezavehi, S., Avgeriou, P., Weyns, D., et al., 2017. A classification framework of uncertainty in architecture-based self- adaptive systems with multiple quality requirements , 33.

Nick, T.G., Campbell, K.M., 2007. Logistic regression. Topics in biostatistics , 273–301.

Niehues, N., Arp, B., Hüller, T., Schwickerath, F., Boltz, N., Hahner, S., 2024. Integrating security-enriched data flow diagrams into architecture-based confidentiality analysis. Gesellschaft für Informatik e.V. https://dl.gi.de/handle/20.500.12116/45546.

Niehues, N., Hahner, S., Heinrich, R., 2025. An architecture-based approach to mitigate confidentiality violations using machine learning. In: Proceedings of the 22Nd IEEE International Conference on Software Architecture (ICSA), p. 12. https://doi.org/10.5445/IR/1000178568

Niehues, N., Hahner, S., Heinrich, R. (2025). Supplementary material for “Mitigation strategies for confidentiality violations in software architecture using ranked feature importance”.

Peldszus, S., Tuma, K., Struber, D., Jurjens, J., Scandariato, R., 2019. Secure data-flow compliance checks between models and code based on automated mappings. IEEE, pp. 23–33. https://doi.org/10.1109/MODELS.2019.00-18

Perez-Palacin, D., Mirandola, R., 2014. Uncertainties in the modeling of self-adaptive systems: a taxonomy and an example of availability evaluation. In: Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering. Association for Computing Machinery, pp. 3–14. https://doi.org/10.1145/2568088.2568095

Ramirez, A.J., Jensen, A.C., Cheng, B. H.C., et al., 2012. A taxonomy of uncertainty for dynamically adaptive systems. In: 2012 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), pp. 99–108. ISSN: 2157–2321. https://doi.org/10.1109/SEAMS.2012.6224396

Runeson, P., Höst, M., 2009. Guidelines for conducting and reporting case study research in software engineering 14 (2), 131.

Schneider, S., Ferreyra, N. E.D., Quéval, P.-J., Simhandl, G., Zdun, U., Scandariato, R., et al., 2024. How dataflow diagrams impact software security analysis: an empirical experiment. 2401.04446 [cs]. http://arxiv.org/abs/2401.04446.

Schneider, S., Özen, T., Chen, M., Scandariato, R., et al., 2023. Microsecend: a dataset of security-enriched dataflow diagrams for microservice applications. In: 2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR), pp. 125–129. ISSN: 2574–3864. https://doi.org/10.1109/MSR59073.2023.00030

Seifermann, S., Heinrich, R., Werle, D., Reussner, R., et al., 2022. Detecting violations of access control and information flow policies in data flow diagrams 184, 111138. https://doi.org/10.1016/j.jss.2021.111138

Seifermann, S., Walter, M., Hahner, S., Heinrich, R., Reussner, R., et al., 2021. Identifying confidentiality violations in architectural design using palladio. In: Companion Proceedings of the 15th European Conference on Software Architecture (ECSA-C). CEUR Workshop Proceedings, pp. 1–4.

Sobhy, D., Bahsoon, R., Minku, L., Kazman, R., et al., 2021. Evaluation of software architectures under uncertainty: a systematic literature review , 50.

Steenhoek, B., Gao, H., Le, W., et al., 2024. Dataflow analysis-inspired deep learning for efficient vulnerability detection. In: Proceedings of the 46th IEEE/ACM International Conference on Software Engineering. ACM, pp. 1–13. https://doi.org/10.1145/3597503.3623345

Su, X., Yan, X., Tsai, C.-L., 2012. Linear regression. Wiley Interdiscip. Rev. Comput. Stat. 4 (3), 275–294.

Troya, J., Moreno, N., Bertoa, M.F., Vallecillo, A., et al., 2021. Uncertainty representation in software models: a survey 20 (4), 1183–1213. https://doi.org/10.1007/s10270-020-00842-1

Tuma, K., Scandariato, R., Balliu, M., 2019. Flaws in flows: unveiling design flaws via information flow analysis. In: 2019 IEEE International Conference on Software Architecture (ICSA). IEEE, pp. 191–200. https://doi.org/10.1109/ICSA.2019.00028

Union, C. o.E., 2016. REGULATION (EU) 2016/679 (general data protection regulation). https://eur-lex.europa.eu/eli/reg/2016/679/2016-05-04.

Walker, W.E., Harremoës, P., Rotmans, J., Van Der Sluijs, J.P., Van Asselt, M. B.A., Janssen, P., Krayer von Krauss, M.P., 2003. Defining uncertainty: a conceptual basis for uncertainty management in model-based decision support. Integrated Asse. 4 (1), 5–17.

Walter, M., Hahner, S., Seifermann, S., Bures, T., Hnetynka, P., Pacovsky, J., Heinrich, R., et al., 2022a. Architectural optimization for confidentiality under structural uncertainty. In: Software Architecture. Springer International Publishing, pp. 309–332. https://doi.org/10.1007/978-3-031-15116-3_14

Walter, M., Heinrich, R., Reussner, R., 2022b. Architectural attack propagation analysis for identifying confidentiality issues. In: 2022 IEEE 19th International Conference on Software Architecture (ICSA). Institute of Electrical and Electronics Engineers (IEEE), p. 12 S. https://doi.org/10.1109/ICSA53651.2022.00009

Walter, M., Heinrich, R., Reussner, R., et al., 2023. Architecture-based attack path analysis for identifying potential security incidents. In: Tekinerdogan, B., Trubiani, C., Tibermacine, C., Scandurra, P., Cuesta, C.E. (Eds.), Software Architecture. Springer Nature Switzerland, pp. 37–53. https://doi.org/10.1007/978-3-031-42592-9_3

Weyns, D., 2020. An introduction to self-adaptive systems: A contemporary software engineering perspective. John Wiley & Sons. https://doi.org/10.1002/9781119574910.

Weyns, D., Calinescu, R., Mirandola, R., Tei, K., Acosta, M., Bennaceur, A., Boltz, N., Bures, T., Camara, J., Diaconescu, A., Engels, G., Gerasimou, S., Gerostathopoulos, I., Getir Yaman, S., Grassi, V., Hahner, S., Letier, E., Litoiu, M., Marsso, L., Musil, A., Musil, J., Nunes Rodrigues, G., Perez-Palacin, D., Quin, F., Scandurra, P., Vallecillo, A., Zisman, A., et al., 2023. Towards a research agenda for understanding and managing uncertainty in self-adaptive systems 48 (4), 20–36. https://doi.org/10.1145/3617946.3617951

Whittle, J., Sawyer, P., Bencomo, N., Cheng, B. H.C., Bruel, J.-M., 2009. Relax: incorporating uncertainty into the specification of self-adaptive systems. In: 2009 17th IEEE International Requirements Engineering Conference. IEEE, pp. 79–88.

Williams, B., Onsman, A., Brown, T., 2010. Exploratory factor analysis: a five-step guide for novices. Australasian journal of paramedicine 8, 1–13.

Xu, T., Zhou, Y., 2015. Systems approaches to tackling configuration errors: a survey. ACM Comput. Surv. 47 (4), 70:1–70:41. https://doi.org/10.1145/2791577

Yong, A.G., Pearce, S., et al., 2013. A beginner's guide to factor analysis: focusing on exploratory factor analysis. Tutor. Quant. Methods Psychol. 9 (2), 79–94.