N. Niehues, S. Hahner, and R. Heinrich, "An Architecture-Based Approach to Mitigate Confidentiality Violations Using Machine Learning", in 2025 IEEE 22nd International Conference on Software Architecture (ICSA), IEEE, 2025, accepted, to appear.

An Architecture-Based Approach to Mitigate Confidentiality Violations Using Machine Learning

Nils Niehues
Karlsruhe Institute of Technology
Karlsruhe, Germany
nils.niehues@kit.edu

Sebastian Hahner
Karlsruhe Institute of Technology
Karlsruhe, Germany
sebastian.hahner@kit.edu

Robert Heinrich
Karlsruhe Institute of Technology
Karlsruhe, Germany
robert.heinrich@kit.edu

***Abstract*—Today's software systems have become increasingly connected and complex, requiring comprehensive analysis to ensure quality properties like confidentiality. Architecture-based confidentiality analysis enables the early identification of confidentiality violations to counter data breaches effectively. However, uncertainty within the software system and its environment hinders the precise and comprehensive analysis of software architectures. Furthermore, the complexity of both architectural models and uncertainties and their outcomes impede automated model repair due to combinatorial explosion. Ultimately, software architects must manually address all confidentiality violations, which is both bothersome and error-prone. Although existing approaches can identify confidentiality violations due to uncertainty, they fall short of mitigating their effects. In this paper, we address this by utilizing machine learning in the confidentiality analysis both to evaluate the criticality of identified violations and to automatically repair them. This bridges the gap between analysis and mitigation, thereby effectively supporting software architects. Evaluation results show that logistic regression provides the best ranking of the importance of uncertainty sources. Combined with incremental testing, our approach outperforms the state of the art and achieves up to a 60-fold reduction in runtime.**

***Index Terms*—software architecture, confidentiality, uncertainty, machine learning, data flow analysis**

I. Introduction

In today's world, software systems are increasingly interconnected and complex, requiring thorough and early analysis to ensure essential quality attributes, such as confidentiality. Confidentiality ensures that data is "only accessible and interpretable by authorized users in a specific context of use" [34]. Unpredictable factors, such as changes in user behavior and system environment, continuously evolve software systems [43]. Such uncertainty make it difficult for developers to maintain confidentiality throughout a system's lifecycle. Furthermore, addressing issues later in the development process increases costs [9, 25], making it important to address these uncertainties at the design stage.

Architecture-based confidentiality analysis is a proactive approach, allowing software architects to identify potential confidentiality violations before they manifest in the deployed system. Data flow and architectural diagrams help software architects detect uncertainty sources that may impact confidentiality early [55], as many uncertainty sources can already be addressed at design time [29]. For instance, uncertainties related to the deployment location of servers or whether encryption is performed can influence data flow and compliance with confidentiality requirements. Although these approaches help identify uncertainty sources, software architects face challenges in assessing their impact on confidentiality requirements due to the many forms of uncertainty and the complexity of modern software systems [26]. The Architecture-Based and Uncertainty-Aware Confidentiality Analysis (ABUNAI) approach [27] can reveal confidentiality violations within the system's design due to uncertainty. For example, if the location of a server processing personal data is uncertain, this can lead to potential violations of regulations, such as the GDPR [20].

While existing analyses can pinpoint confidentiality violations, they fall short of providing solutions to mitigate them [50, 57]. Software architects must manually address each identified confidentiality violation, a task that is labor-intensive and prone to human error, particularly in large-scale systems [71]. Furthermore, the exponential growth of possible combinations of uncertainty scenarios poses a significant challenge for automated solutions, as trying out all possible configurations quickly becomes computationally prohibitive, an issue known from the field of design space exploration [39, 40].

This paper introduces a machine-learning approach to mitigate confidentiality violations, addressing these limitations with two major contributions:

- **C1** A ranking algorithm for the relevance of uncertainties on confidentiality violations examining 18 combinations of machine learning techniques and aggregation strategies.
- **C2** An automatic mitigation approach that utilizes the ranking algorithm to efficiently provide model variations without confidentiality violation.

By ranking uncertainties based on their potential to cause confidentiality breaches and subsequently testing combinations of uncertainty scenarios, our approach identifies and resolves confidentiality risks in an efficient manner. This supports software architects in maintaining confidentiality requirements. Our experiments on multiple evaluation scenarios show that a combination of logistic regression with exponential decay provides the most accurate ranking of uncertainties. Combined with incremental testing, notably reduces mitigation runtime compared to the start of the art [27]. In our evaluation, this improvement cuts the time from 2 hours down to just 2 minutes—a reduction of up to 60 times. Even on average, our approach still outperforms the state of the art.

The paper is structured as follows. Section II describes the foundations for this work and Section III introduces a running example. In Section IV, we provide an overview of the approach, setting the stage for the uncertainty ranking in Section V and the mitigation of violations in Section VI. We evaluate our work in Section VII, present related work in Section VIII, and conclude the paper in Section IX.

II. 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.

A. DFDs and confidentiality analysis

DeMarco [18] 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 [42, 55].

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.* [57] 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.* [12] developed an improved DFD-based model. Their method leverages Transposed Flow Graphs (TFGs), which are 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 [12]. As such an analysis either finds confidentiality violations or does not, we consider confidentiality a strictly binary attribute.

B. Uncertainty in software architecture

Uncertainty is defined as "the state, even partial, of deficiency of information related to, understanding or knowledge of, an event, its consequence, or likelihood" [34]. 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 [63]. 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 thus effectively be addressed using architectural approaches [29].

Hahner *et al.* [29] 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 [27] model 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.

C. 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 [33, 72]. Principal Component Analysis (PCA) reduces dimensionality by transforming variables into orthogonal components, revealing primary variance sources [1]. Random Forests (RF) improves predictive accuracy through an ensemble of decision trees, especially in high-dimensional data [14]. Linear Regression (LR) and Logistic Regression (LGR) model relationships between inputs and outcomes, offering interpretable coefficients and binary classification, respectively [46, 60]. Linear Discriminant Analysis (LDA) enhances class separation by projecting data into lower-dimensional space [5].

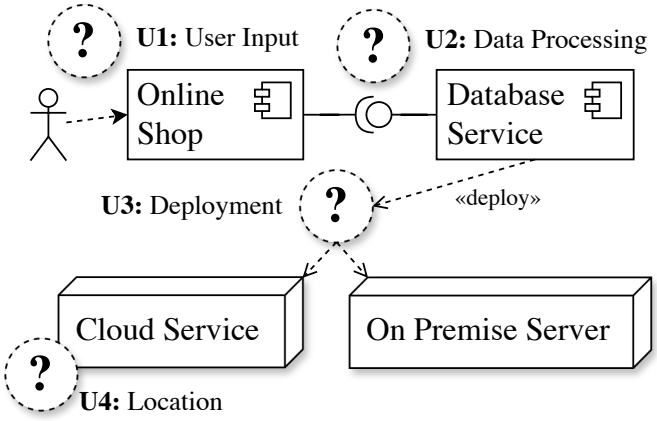


Figure 1: Running example of a simple online shop

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

Table I: All possible combinations for the given uncertainties

III. Running Example

We illustrate the application and impact of our mitigation approach with a simplified model of an online shop. Figure 1 shows the architecture model consisting of the user, two software components, and two data storages and has been used in similar work [31]. 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:

$$hasDataLabel(Personal) \wedge \neg hasNodeLabel(nonEU) \quad (1)$$
$$hasDataLabel(Personal) \wedge hasDataLabel(Sanizited) \quad (2)$$

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 I, 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 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.

IV. Overview of the Approach

This paper introduces a novel approach for analyzing and modifying DFDs to ensure compliance with confidentiality constraints by addressing uncertainties within the model. This section serves as an overview of our process, displayed in Figure 2, which centers on identifying and mitigating uncertainties that impact confidentiality violations. Our approach systematically identifies and configures valid scenarios in the input model. By only changing the uncertainties necessary to make the model compliant with confidentiality requirements, our approach preserves as much of the original model as possible while reducing the analysis runtime.

In Contribution **C1**, 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 process 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

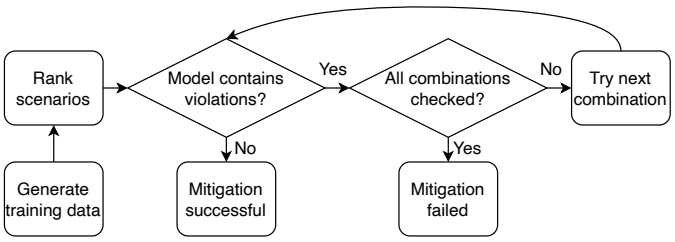


Figure 2: Simplified pipeline of the approach

based on their potential to cause violations. This ranking helps prioritize uncertainties in subsequent steps.

In Contribution **C2**, 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.

V. Ranking Uncertainties Based on Constraint Violations

This section introduces contribution **C1**, an approach for ranking uncertainties by analyzing their impact on constraint violations through machine learning techniques. We begin with methods for generating training data tailored for ranking purposes, then techniques for assessing the importance of uncertainties, and finally, a strategy for aggregating rankings across multiple constraints. Each step leverages machine learning to systematically prioritize uncertainties with the greatest potential impact on confidentiality or other constraints.

A. Generating Training Data for Ranking Uncertainties

The process begins with generating training data for uncertainty ranking using TFGs, each representing a unique combination of uncertainty scenarios. ABUNAI checks each TFG to determine whether a confidentiality violation occurs [27]. We translate each TFG and its violation status into categorical data, enabling machine learning models to analyze the conditions that lead to violations.

We generate separate training data for each constraint. Each row corresponds to a TFG, while each column represents a specific uncertainty scenario. When a TFG does not flow through all uncertainty sources, we use placeholder values for missing uncertainties to maintain a consistent tabular format. In the running example case, the cloud server's location is irrelevant if we store data on the on-premise server. This structured approach allows for machine learning models to process data even when some uncertainties are absent in specific scenarios.

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

Table II: Training data for Figure 1, where **U** stands for uncertainty, **C** for constraint, **S** for scenario and **-** for irrelevant

To balance the dataset, we include both TFGs that cause confidentiality violations and those that do not. This improves the model's ability to identify critical uncertainties by reducing the noise from irrelevant configurations. The running example from Section III consists of 4 uncertainty sources, with 2 scenarios each and 2 constraints, which gets transformed into the categorical training data displayed in Table II.

B. Ranking Uncertainties for Importance

Using machine learning, we can prioritize uncertainties by analyzing how their different scenarios influence constraint violations. Both unsupervised and supervised learning approaches are employed to derive rankings.

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 retaining the most critical variables[1, 70]. PCA prioritizes uncertainties explaining 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 subtle patterns across TFGs that might influence constraint violations.

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 by violation status [44]. Each technique assesses the importance

Algorithm 1 Ranking Unsupervised

```
function RANKUNSUPERVISED(data_rows)
    ranking ← {}
    result ← CREATECOMPONENTS(data_rows)
    components ← result.Components
    loadings ← components.Loadings
    for loading ∈ loadings do
        continue for Constraint violated column
        ranking[loading.Name] ← SUM(loading.Values)
    end for
    return SORTDICTDESCENDING(ranking)
end function
```

of each 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 [5]. In contrast, Random Forests measure variable importance through mean decreases in impurity across decision trees, capturing both individual and combined effects of uncertainties [14, 22]. Algorithm 2 illustrates the process for deriving feature importance from each model, providing a foundation for ranking uncertainties by their contribution to constraint violations.

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
```

To illustrate the ranking result, we rank the training data for the running example shown in Table II with logistic regression. Table III shows the importance of uncertainties for the individual constraints. In Table IIIa shows that **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 IIIb shows that **U1** and **U2** are equally important for constraint 2 because only combinations of these two can lead to a sanitization violation.

Uncertainty	Importance
U4	0.6307
U1	0.5836
U3	0.5713
U2	0.5000

(a) Constraint 1

Uncertainty	Importance
U1	0.6250
U2	0.6250
U3	0.5000
U4	0.5000

(b) Constraint 2

Table III: Individual importance of uncertainties for the running example

C. 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. We aggregate rankings from individual constraints to construct an overall ranking, focusing only on those globally impacting confidentiality violations. We normalize each constraint-specific ranking by summing and dividing by the total, ensuring all scores range from 0 to 1. This normalization maintains comparability across constraints with varying levels of severity in violations. Several aggregation methods are evaluated, each offering unique benefits:

1) Simple Summation: This method sums the normalized scores from each constraint-specific ranking, producing a cumulative score. While straightforward, this approach elevates uncertainties that are only moderately important across multiple constraints but are not critical to any specific violation.

2) Exponential Decay: Applying e^{-r} to the summed-up rank r of each uncertainty reduces the impact of lower-ranked uncertainties. This approach emphasizes the top-ranked uncertainties from each constraint, providing a clearer focus on the most significant factors.

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. While this approach risks missing lower-ranked yet impactful uncertainties, it reduces noise from uncertainties with minimal contributions to constraint violations.

This approach unifies constraint-specific rankings to identify the uncertainties with the greatest potential for mitigating confidentiality or compliance violations across complex models. While the respective optimal aggregation strategy depends on the structure of the DFD and 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 shown in Table III aggregated using exponential decay. The results are shown in Table IV and confirm that **U1** is the most important one because it can lead to violations of both constraints.

Uncertainty	Importance
U1	1.3678
U4	1.0000
U2	0.3678
U3	0.1353

Table IV: Aggregated importance of uncertainties for the running example

VI. MITIGATING CONFIDENTIALITY VIOLATIONS

We mitigate confidentiality violations by configuring the software architecture to use a combination of uncertainty scenarios that comply with confidentiality requirements. Our process 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 process terminates. If not, the approach continues, using other uncertainties in the ranking until no further options are available. 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.

A. Depth-First-Search

The simplest approach involves trying all combinations of uncertainties ranked by appearance in the TFGs, allowing a straightforward analysis without additional overhead. This depth-first-search method 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 approach serves as a baseline for our evaluation.

B. Incremental Increase in Uncertainties

In this method, 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 approach 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 approaches.

C. 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 approach can improve efficiency by reducing redundant combination checks. In this method, 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 process 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 process. This strategy performs efficiently when there are many violation 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.

D. 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 process 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 ∪ m_j
9:                  if mergedModel satisfies scenarios then
10:                     u ← irrelevant
11:                 end if
12:                 newModels ← newModels ∪ mergedModel
13:                 newModels ← newModels \m_i \ 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.

VII. 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. This section presents our evaluation plan, design, and results and discusses threats to validity.

A. Goal, Questions, and Metrics

We use a Goal-Question-Metric (GQM) plan [6, 7] to structure the evaluation. To enhance validity, we align our plan with related work [28, 30] and use well-known metrics [36].

Our **Goal** is to evaluate the quality of our mitigation approach compared to existing approaches [28, 30]. 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 proposed mitigation strategies?

Question **Q1** asks about 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 II. Furthermore, we consider the three aggregation strategies introduced in Section V, 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" [36]. 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 [28]. Put simply, we investigate how many irrelevant uncertainties were ranked among all relevant uncertainties, where lower is better.

Question **Q2** evaluates 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.

Last, Question **Q3** asks about the scalability of the mitigation. We consider the mitigation strategies that determine how uncertainties are selected for the mitigation, i.e., incremental increase (INCREASE) and fixed subsets splitting the relevant uncertainties into two halves (HALF), or four quarters (QUARTER), see Section VI. We compare these strategies against each other and against Depth-First-Search (DFS), 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.

B. 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 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.

a) Spring Boot: Kothagal [38] created this open-source microservice architecture, and Schneider *et al.* [54] derived its DFD representation. This model 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 [48]:

$$hasDataLabel(entrypoint) \implies hasDataLabel(encrypted\_connection)$$
$$hasNodeLabel(internal) \implies hasDataLabel(encrypted\_connection)$$
$$hasNodeLabel(local\_logging) \implies hasNodeLabel(internal)$$

b) Tap and Eat: Ferrater [23] created this open-source architecture, and Schneider *et al.* [54] derived its DFD. This model 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 [48]:

$$hasNodeLabel(internal) \implies hasDataLabel(auth\_request)$$
$$hasNodeLabel(login\_attempts\_reg) \implies hasNodeLabel(auth\_server)$$
$$hasDataLabel(entrypoint) \implies hasDataLabel(encrypted\_connection)$$
$$hasNodeLabel(local\_logging) \implies hasNodeLabel(internal)$$

c) Online Banking: The last model is our own creation and displays an online banking architecture consisting 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:

$$hasNodeLabel(nonEU) \implies \neg hasDataLabel(Personal)$$
$$hasNodeLabel(Processable) \implies \neg hasDataLabel(Encrypted)$$
$$hasNodeLabel(Develop) \implies \neg hasDataLabel(Personal)$$

C. Design

To evaluate the precision of our ranking, we created reference rankings by manually examining the evaluation scenarios and identifying the relevant uncertainties. We consider these uncertainties to be the true positives. Afterward, we executed our mitigation approach 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 [27]. 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.

D. Results and Discussion

1) Ranking of uncertainties: We evaluated each uncertainty ranker with every aggregation strategy, resulting in 18 Precision@K scores per model. The results for the Spring Boot model are displayed in Figure 3a, for the Tap and Eat model in Figure 3b, and for the Online Banking model in Figure 3c. Each bar represents a Precision@K value ranging from 0 to 1, with 1 indicating a perfect rating.

The results demonstrate that supervised rankers consistently outperformed unsupervised ones across all evaluation scenarios, with Logistic Regression and Random Forest emerging as the top-performing methods. The Exponential and Top3 aggregation strategies combined with Logistic Regression consistently produce perfect rankings across all models. This performance suggests that supervised rankers align well with the task, effectively capturing the relevant uncertainties.

In contrast, unsupervised rankers, such as EFA and PCA, underperformed, especially in models with high variance among uncertainties. This likely stems from their reliance on shared variance, which can introduce noise when unrelated variables correlate with the target constraints.

The Exponential and Top3 aggregation methods excelled by isolating top-ranked uncertainties directly impacting violations, focusing on the most critical uncertainties without diluting from lower-ranked factors. Conversely, the Sum aggregation method performed the worst across all models, likely due to its even distribution of importance, which introduces random noise and reduces precision.

In summary, by combining logistic regression with exponential decay, we achieved a perfect ranking of uncertainties across all evaluation scenarios.

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

The results, summarized in Table V, display that prior to mitigation, the Spring Boot model exhibited 3 confidentiality violations, the Tap and Eat model had 9, and the Online Banking Model recorded 4 violations. After applying our mitigation approach, all models demonstrated zero confidentiality violations. These findings confirm the effectiveness of our approach in mitigating confidentiality violations.

3) Scalability: To evaluate the scalability of our mitigation approach, we modified the models by duplicating or removing subgraphs. This adjustment yielded models with uncertainty counts ranging from 7 to 22. We further maintained a proportional distribution of relevant to total uncertainties to preserve model behavior and structure. This variation provides a broader set of data points for evaluating scalability. We executed the mitigation approach multiple times for each uncertainty level, recording the average runtime in milliseconds on a logarithmic scale. The results are shown in Figure 4.

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

Table V: Confidentiality violations identified before and after mitigation

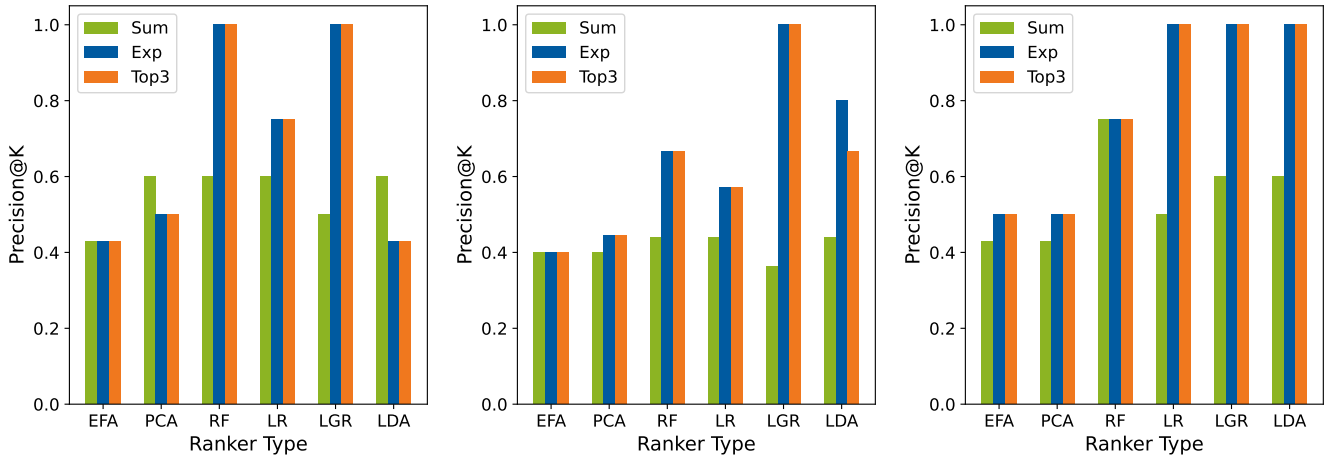
The scaled Spring Boot and Online Banking models demonstrate consistent results. The Depth-First Search baseline scales exponentially, at a rate of 2^n for n uncertainties, while our approach achieves a more efficient scaling rate of approximately $2^{\frac{n}{2}}$. This efficiency is attributed to the effective ranking of uncertainties, which allows our method to consider only about half the scenario combinations necessary to mitigate confidentiality violations. Notably, a fixed batch size that splits uncertainties into two subsets yielded optimal results in these scenarios, which stems from the high number of relevant uncertainties.

In contrast, the Tap and Eat model exhibits a different scaling pattern. The Depth-First Search baseline continues to scale with 2^n for n uncertainties, yet our fixed batch size approach is less efficient here due to the lower count of relevant uncertainties. However, the iterative incremental mitigation strategy outperforms other approaches, scaling at an approximate rate of $2^{\frac{n}{2}}$. This result aligns with expectations, as a lower number of relevant uncertainties benefits from incremental adjustments rather than fixed batches.

Overall, our mitigation approach introduces overhead, resulting in the baseline being faster on smaller models. As we go over 15 uncertainties, our mitigation approaches consistently outperform the baseline with up to 60 times faster mitigation for the Online Banking model at 22 uncertainties.

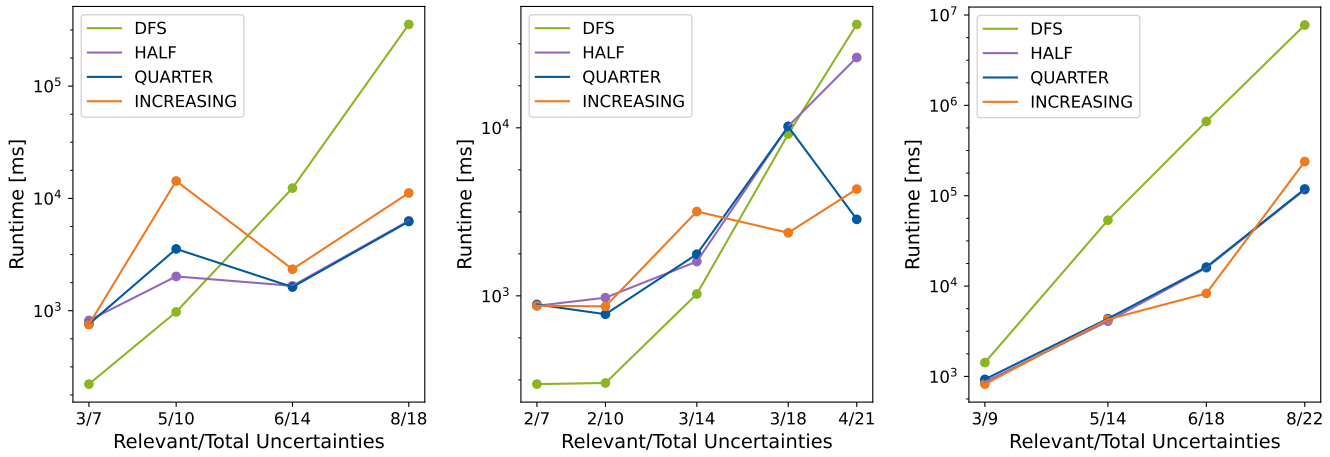
E. Threats to validity

We structure the discussion of threats to validity based on the guidelines by Runeson and Höst [53]. Regarding the *internal validity*, one threat is the accuracy of the model-based confidentiality analysis [30] 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. Regarding the *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.* [54] being derived from real-world open-source projects. To ensure *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



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

Figure 3: Precision@K for different uncertainty rankers and aggregation strategies



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

Figure 4: Runtime scalability for different mitigation strategies

reliability of our results and minimize the bias in annotating our models. We did not consider hypothesis testing in our evaluation because 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 nine runs per measurement. Last, we provide supplemental material as data set [47] to address the availability of evolution artifacts in software architecture research [37]. This includes all code artifacts, results, and instructions on reproducing them.

VIII. 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 [32, 67, 68]. Second, approaches to confidentiality analysis at design time that enable the early identification of confidentiality violations [10, 12, 13, 57], also while considering uncertainty [28, 30]. Third, using machine learning to mitigate security issues [2, 41, 59]. 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.* [61] 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.* [4] support this view by presenting modeling dimensions of self-adaptive systems, as does the recently proposed OMG *PSUM* standard [49]. Researchers have proposed numerous classifications [45, 51, 52] to investigate the nature of uncertainty, including some focused on security-related fields like access control [15] and confidentiality [29]. Comprehensive frameworks have been

researched to investigate uncertainty in software systems, e.g., *RELAX* [69], Rainbow [24], PerOpteryx [40], ArcheOpterix [3], GuideArch [19], and DeTUM [21]. Here, Hezavehi *et al.* [32] and Sobhy *et al.* [58] 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 [30, 64] to identify and repair confidentiality violations under uncertainty. Researchers have highlighted the challenge of creating comprehensive end-to-end approaches [68]. Here, especially considering multiple uncertainty sources and their interactions is considered to be challenging [16, 17]. 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 [12, 56, 57], or architecture-based access control analysis [64, 65]. Furthermore, broader approaches to model-based security analysis, e.g. UMLsec [35], or SecDFD [50, 62]. 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 [66], by tracing uncertainty in data flow diagrams [30], or by combining data flow analysis with fuzzy inference to represent uncertainty [11]. Hahner *et al.* [28] 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.* [2] 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.* [41], who extract security-related features from source code to use in models like decision trees and random forests. Another recently proposed approach by [59] 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 [8, 31] 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 [55]. Furthermore, only considering source code in the analysis limits the applicability at design time, which is required for early mitigation to minimize costs [9]. 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.

IX. Conclusion

In this paper, we introduced an approach to mitigate confidentiality violations in software architectures utilizing machine learning. Specifically, we contributed **C1**, a method for ranking critical uncertainties that lead to confidentiality violations by leveraging Data Flow Diagrams combined with machine learning techniques, and **C2**, an automated framework for mitigating confidentiality violations in software architectures.

Our exhaustive experiments demonstrate that logistic regression with exponential decay provides the most accurate ranking of uncertainties with an ideal Precision@K score of 1. This accurate ranking and incremental scenario testing notably reduce the mitigation runtime. In our evaluation, this improvement led to a runtime reduction of up to 60 times, decreasing the duration from 2 hours to just 2 minutes. Even on average, our approach still outperforms the state of the art. Furthermore, our approach successfully mitigates confidentiality violations in software architectures.

This work bridges the gap between existing analysis frameworks, which primarily focus on identifying confidentiality violations, and an automatic mitigation approach that assists software architects in efficiently addressing confidentiality challenges at design time. Our approach automates uncertainty prioritization and resolution, enhancing the scalability of confidentiality maintenance in complex systems ensuring that data protection measures are more manageable and reliable. This supports software architects in mitigating the negative effects of uncertainty sources that resolve at design time or can at least partially be addressed using the software architecture.

Future work could extend this framework by automatically generating scenarios using machine learning trained on real-world software architectures, which would further reduce the manual effort of software architects. Conducting additional experiments with more complex mitigation strategies, such as clustering uncertainties, could further reduce the analysis runtime. Additionally, we plan to explore combining SAT solvers with fine-grained cost estimation to identify a configuration that satisfies all confidentiality requirements while minimizing the cost of the mitigation.

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

Acknowledgment

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). We like to thank Jonas Koch and Benjamin Arp.

References

[1] H. Abdi and L. J. Williams, "Principal component analysis," *Wiley interdisciplinary reviews: computational statistics*, vol. 2, no. 4, pp. 433–459, 2010.

[2] M. Ahsan *et al.*, "Cybersecurity threats and their mitigation approaches using machine learning—a review," *Journal of Cybersecurity and Privacy*, vol. 2, no. 3, pp. 527–555, 2022.

[3] A. Aleti *et al.*, "ArcheOpterix: An extendable tool for architecture optimization of AADL models," in *2009 ICSE Workshop on Model-Based Methodologies for Pervasive and Embedded Software*, 2009, pp. 61–71.

[4] J. Andersson *et al.*, "Modeling dimensions of self-adaptive software systems," in *Software Engineering for Self-Adaptive Systems*, 2009, pp. 27–47.

[5] S. Balakrishnama and A. Ganapathiraju, "Linear discriminant analysis-a brief tutorial," *Institute for Signal and information Processing*, vol. 18, no. 1998, pp. 1–8, 1998.

[6] V. R. Basili, "Software modeling and measurement: The goal/question/metric paradigm," Tech. Rep., 1992.

[7] V. R. Basili and D. M. Weiss, "A methodology for collecting valid software engineering data," *IEEE Transactions on Software Engineering*, vol. SE-10, no. 6, pp. 728–738, 1984.

[8] M. M. Bersani *et al.*, "A conceptual framework for explainability requirements in software-intensive systems," in *2023 IEEE 31st International Requirements Engineering Conference Workshops (REW)*, 2023, pp. 309–315.

[9] B. Boehm and V. Basili, "Software defect reduction top 10 list," *Computer*, vol. 34, no. 1, pp. 135–137, 2001, Conference Name: Computer.

[10] N. Boltz, M. Walter, and R. Heinrich, "Context-based confidentiality analysis for industrial IoT," in *SEAA*, 2020, pp. 589–596.

[11] N. Boltz *et al.*, "Handling environmental uncertainty in design time access control analysis," in *SEAA*, 2022, pp. 382–389.

[12] N. Boltz *et al.*, "An extensible framework for architecture-based data flow analysis for information security," in *European Conference on Software Architecture*, Springer, 2023, pp. 342–358.

[13] N. Boltz *et al.*, "Modeling and analyzing zero trust architectures regarding performance and security," in *Software Architecture*, 2024, pp. 253–269.

[14] L. Breiman, "Random forests," *Machine learning*, vol. 45, pp. 5–32, 2001.

[15] T. Bures *et al.*, "Capturing dynamicity and uncertainty in security and trust via situational patterns," in *Leveraging Applications of Formal Methods, Verification and Validation: Engineering Principles*, 2020, pp. 295–310.

[16] J. Camara *et al.*, "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*, 7, 2024, pp. 37–43.

[17] J. Cámara *et al.*, "The uncertainty interaction problem in self-adaptive systems," *SoSyM*, vol. 21, no. 4, pp. 1277–1294, 1, 2022.

[18] T. DeMarco, "Structured analysis and system specification. yourdon," *Inc., New York, New York*, 1978.

[19] N. Esfahani, S. Malek, and K. Razavi, "GuideArch: Guiding the exploration of architectural solution space under uncertainty," in *ICSE*, ISSN: 1558-1225, 2013, pp. 43–52.

[20] C. of European Union. "REGULATION (EU) 2016/679 (general data protection regulation)." (2016), [Online]. Available: https://eur-lex.europa.eu/eli/reg/2016/679/2016-05-04 (visited on 01/19/2021).

[21] M. Famelis and M. Chechik, "Managing design-time uncertainty," *Software & Systems Modeling*, vol. 18, no. 2, pp. 1249–1284, 1, 2019.

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

[23] J. Ferrater, *Tap-and-eat-microservices*, Accessed: 2024-11-16, 2024. [Online]. Available: https://github.com/jferrater/Tap-And-Eat-MicroServices.

[24] D. Garlan *et al.*, "Rainbow: Architecture-based self-adaptation with reusable infrastructure," *Computer*, vol. 37, no. 10, pp. 46–54, 2004.

[25] M. Glinz, "Requirements engineering i," *Nicht funktionale Anforderungen. Universität Zürich, Institut für Informatik, Zürich*, 2006.

[26] S. Hahner, "Dealing with uncertainty in architectural confidentiality analysis," in *Proceedings of the Software Engineering 2021 Satellite Events*, 2021, pp. 1–6.

[27] S. Hahner, "Architecture-based and uncertainty-aware confidentiality analysis," Dissertation, Karlsruhe Institute of Technology (KIT), 2024.

[28] S. Hahner, R. Heinrich, and R. Reussner, "Architecture-based uncertainty impact analysis to ensure confidentiality," in *2023 IEEE/ACM 18th Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, ISSN: 2157-2321, 2023, pp. 126–132.

[29] S. Hahner *et al.*, "A classification of software-architectural uncertainty regarding confidentiality," in *E-Business and Telecommunications*, 2023, pp. 139–160.

[30] S. Hahner *et al.*, "Model-based Confidentiality Analysis under Uncertainty," in *2023 IEEE 20th International Conference on Software Architecture Companion (ICSA-C)*, 2023, pp. 256–263.

[31] S. Hahner *et al.*, "Arc³n: 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)*, 2024.

[32] S. M. Hezavehi *et al.*, "Uncertainty in self-adaptive systems: A research community perspective," vol. 15, no. 4, 2021.

[33] D. Hooper, "Exploratory factor analysis," 2012.

[34] ISO, "ISO/IEC 27000:2018(e) information technology – security techniques – information security management systems – overview and vocabulary," Standard, 2018.

[35] J. Jürjens, "UMLsec: Extending UML for secure systems development," in *UML 2002 — The Unified Modeling Language*, red. by G. Goos, J. Hartmanis, and J. van Leeuwen, vol. 2460, Series Title: Lecture Notes in Computer Science, 2002, pp. 412–425.

[36] P. Kar, H. Narasimhan, and P. Jain, "Surrogate functions for maximizing precision at the top," in *Proceedings of the 32nd International Conference on Machine Learning*, vol. 37, 2015, pp. 189–198.

[37] M. Konersmann *et al.*, "Evaluation methods and replicability of software architecture research objects," in *ICSA*, 2022, pp. 157–168.

[38] K. Kothagal, *Spring boot microservices workshop*, Accessed: 2024-11-16, 2024. [Online]. Available: https://github.com/koushikkothagal/spring-boot-microservices-workshop.

[39] A. Koziolek, "Automated Improvement of Software Architecture Models for Performance and Other Quality Attributes," Ph.D. dissertation, 2011. DOI: 10.5445/IR/1000024955.

[40] A. Koziolek, H. Koziolek, and R. Reussner, "PerOpteryx: Automated application of tactics in multi-objective software architecture optimization," in *ISARCS*, 20, 2011, pp. 33–42.

[41] J. Kronjee, A. Hommersom, and H. Vranken, "Discovering software vulnerabilities using data-flow analysis and machine learning," in *Proceedings of the 13th International Conference on Availability, Reliability and Security*, 27, 2018, pp. 1–10.

[42] P. G. Larsen, N. Plat, and H. Toetenel, "A formal semantics of data flow diagrams," *Formal aspects of Computing*, vol. 6, pp. 586–606, 1994.

[43] M. Lehman and J. C. Fernáandez-Ramil, "Software evolution," *Software evolution and feedback: Theory and practice*, p. 7, 2006.

[44] P. Linardatos, V. Papastefanopoulos, and S. Kotsiantis, "Explainable ai: A review of machine learning interpretability methods," *Entropy*, vol. 23, no. 1, p. 18, 2020.

[45] S. Mahdavi-Hezavehi, P. Avgeriou, and D. Weyns, "A classification framework of uncertainty in architecture-based self- adaptive systems with multiple quality requirements," *Managing Trade-Offs in Adaptable Software Architectures*, p. 33, 2017.

[46] T. G. Nick and K. M. Campbell, "Logistic regression," *Topics in biostatistics*, pp. 273–301, 2007.

[47] N. Niehues, S. Hahner, and R. Heinrich, *Supplementary material for "an architecture-based approach to mitigate confidentiality violations using machine learning"*, 2025. DOI: 10.5281/zenodo.14723725.

[48] N. Niehues *et al.*, "Integrating security-enriched data flow diagrams into architecture-based confidentiality analysis," 2024.

[49] Object Management Group, *Precise semantics for uncertainty modeling (PSUM), version 1.0 beta 2*, 2024. [Online]. Available: https://www.omg.org/spec/PSUM/1.0/Beta2/PDF.

[50] S. Peldszus *et al.*, "Secure data-flow compliance checks between models and code based on automated mappings," presented at the 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS), 2019, pp. 23–33.

[51] D. Perez-Palacin and R. Mirandola, "Uncertainties in the modeling of self-adaptive systems: A taxonomy and an example of availability evaluation," in *ICPE*, 2014, pp. 3–14.

[52] A. J. Ramirez, A. C. Jensen, and B. H. C. Cheng, "A taxonomy of uncertainty for dynamically adaptive systems," in *2012 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, ISSN: 2157-2321, 2012, pp. 99–108.

[53] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical software engineering*, vol. 14, no. 2, p. 131, 2009.

[54] S. Schneider *et al.*, "microSecEnD: A dataset of security-enriched dataflow diagrams for microservice applications," in *2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR)*, ISSN: 2574-3864, 2023, pp. 125–129.

[55] S. Schneider *et al.*, *How dataflow diagrams impact software security analysis: An empirical experiment*, 9, 2024. [Online]. Available: http://arxiv.org/abs/2401.04446 (visited on 01/15/2024).

[56] S. Seifermann *et al.*, "Identifying confidentiality violations in architectural design using palladio," in *ECSA-C*, 2021, pp. 1–4.

[57] S. Seifermann *et al.*, "Detecting violations of access control and information flow policies in data flow diagrams," *JSS*, vol. 184, p. 111 138, 1, 2022.

[58] D. Sobhy *et al.*, "Evaluation of software architectures under uncertainty: A systematic literature review," *ACM TOSEM*, p. 50, 2021.

[59] B. Steenhoek, H. Gao, and W. Le, "Dataflow analysis-inspired deep learning for efficient vulnerability detection," in *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*, 6, 2024, pp. 1–13.

[60] X. Su, X. Yan, and C.-L. Tsai, "Linear regression," *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 4, no. 3, pp. 275–294, 2012.

[61] J. Troya *et al.*, "Uncertainty representation in software models: A survey," *SoSyM*, vol. 20, no. 4, pp. 1183–1213, 1, 2021.

[62] K. Tuma, R. Scandariato, and M. Balliu, "Flaws in flows: Unveiling design flaws via information flow analysis," in *2019 IEEE International Conference on Software Architecture (ICSA)*, 2019, pp. 191–200.

[63] W. E. Walker *et al.*, "Defining uncertainty: A conceptual basis for uncertainty management in model-based decision support," *Integrated assessment*, vol. 4, no. 1, pp. 5–17, 2003.

[64] M. Walter, R. Heinrich, and R. Reussner, "Architectural attack propagation analysis for identifying confidentiality issues," in *ICSA*, 2022, 12 S.

[65] M. Walter, R. Heinrich, and R. Reussner, "Architecture-based attack path analysis for identifying potential security incidents," in *Software Architecture*, 2023, pp. 37–53.

[66] M. Walter *et al.*, "Architectural optimization for confidentiality under structural uncertainty," in *Software Architecture*, 2022, pp. 309–332.

[67] D. Weyns, *An introduction to self-adaptive systems: A contemporary software engineering perspective*. John Wiley & Sons, 2020.

[68] D. Weyns *et al.*, "Towards a research agenda for understanding and managing uncertainty in self-adaptive systems," *ACM SIGSOFT Software Engineering Notes*, vol. 48, no. 4, pp. 20–36, 2023.

[69] J. Whittle *et al.*, "Relax: Incorporating uncertainty into the specification of self-adaptive systems," in *2009 17th IEEE International Requirements Engineering Conference*, IEEE, 2009, pp. 79–88.

[70] B. Williams, A. Onsman, and T. Brown, "Exploratory factor analysis: A five-step guide for novices," *Australasian journal of paramedicine*, vol. 8, pp. 1–13, 2010.

[71] T. Xu and Y. Zhou, "Systems Approaches to Tackling Configuration Errors: A Survey," *ACM Comput. Surv.*, vol. 47, no. 4, 70:1–70:41, 2015.

[72] A. G. Yong, S. Pearce, *et al.*, "A beginner's guide to factor analysis: Focusing on exploratory factor analysis," *Tutorials in quantitative methods for psychology*, vol. 9, no. 2, pp. 79–94, 2013.