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), IEEE/ACM, 2023, doi: 10.1109/SEAMS59076.2023.00026

Architecture-based Uncertainty Impact Analysis to ensure Confidentiality

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

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

Ralf Reussner
Karlsruhe Institute of Technology
Karlsruhe, Germany
ralf.reussner@kit.edu

***Abstract*—Today's software systems are neither built nor operated in isolation and have to adapt to their environment. Uncertainty in the software and its context is inherently unavoidable and should be actively analyzed and managed already at design time. This includes analyzing the impact of uncertainty on a system's quality properties, which quickly becomes critical, e.g., regarding confidentiality. When not handled comprehensively, confidentiality violations can occur due to uncertainty that void previous analysis results. There exist many approaches to classify and handle uncertainty. However, without locating the impact of uncertainty, precise mitigation is often impossible. In this paper, we present an uncertainty impact analysis that shows potential confidentiality violations induced by different uncertainty types like structural, behavioral, or environmental uncertainty. This is achieved by combining software-architectural and data flow-based propagation of uncertainty. Our tool-supported approach is a first step towards predicting the impact of uncertainty without laborious modeling and testing of what-if scenarios. The case study-based evaluation shows that our impact analysis accurately predicts confidentiality violations with a high F1-score of 0.94 while reducing the effort of manual analysis by 82%.**

***Index Terms*—Software Architecture, Data Flow Analysis, Uncertainty, Uncertainty Management, Privacy, Confidentiality**

I. Introduction

Uncertainty has become a significant concern in the field of software engineering [14], particularly regarding its impact on a software's quality properties. This can quickly become critical, e.g., regarding privacy and security-related properties like confidentiality. Confidentiality demands that "information is not made available or disclosed to unauthorized individuals, entities, or processes" [26]. Confidentiality violations can have legal consequences [25] and harm user acceptance [49]. For proactive decision-making as implied by *Privacy by Design* [39] and adaptation to a changing environment, uncertainty should be analyzed and managed as early as possible.

Much is yet unknown about potential uncertainty sources and their effects [17], e.g., in early design due to abstract requirements and open decisions [30], or in systems of systems because of unpredictable behavior and complex dependencies and interactions [31]. Here, uncertainty in a software system (e.g., protocol choices) or its environment (e.g., sensor input) can void assumptions [1]. Design time analyses enable architects to identify potential confidentiality violations early [43] based on ensuring confidentiality requirements [20]. However, they are limited in modeling and analyzing uncertainty [5], [47]. Also, uncertainty is often discussed by its source rather than by its impact [19] which impedes its precise mitigation.

Hezavehi et al. [24] recently found a "lack of systematic approaches for managing uncertainty" [24]. Acosta et al. [1] also propose a propagation step of uncertainty sources to their impact locations prior to mitigation. Although confidentiality analyses exist that take the impact of uncertainty into account [21], the connection to uncertainty sources is still missing. This also becomes visible in a recent study by Troya et al. [46].

In this paper, we present a tool-supported approach for *Uncertainty Impact Analysis* regarding confidentiality. Based on software-architectural modeling, we propagate uncertainty sources through the software system to identify impact locations that subsequently can be analyzed and mitigated accordingly. This approach fills the gap between identifying uncertainty sources and understanding their actual impact on a system's confidentiality by using the information of the software's architecture. To achieve this, we build upon the concept of architecture-based change impact analysis [22]. Such analyses trace changes (e.g., replacing components or altering interfaces) and predict the impact on the overall software system. To analyze the impact of uncertainty on confidentiality, we combine this structural propagation with the propagation along extracted data flows. This enhances the precision of the impact especially regarding confidentiality as "problems tend to follow the data flow, not the control flow" [44]. By calculating the impact of uncertainty, software architects can quickly identify confidentiality issues already at design time without laborious modeling and analysis of architectural variations [21] or what-if scenarios. Our contributions are:

- **C1** A modeling approach of software-architectural uncertainty and its impact on data flow diagrams to analyze confidentiality, presented in Section II.
- **C2** An uncertainty impact analysis that propagates uncertainty based on five distinct propagation algorithms and yields uncertainty impact sets, presented in Section III.

Our evaluation in Section IV is based on a real-world case study using the open-source contact tracing app *Corona Warn App* [35]. The results indicate high accuracy of the calculated impact sets and also reduced manual effort. Section V discusses related work and Section VI concludes this paper.

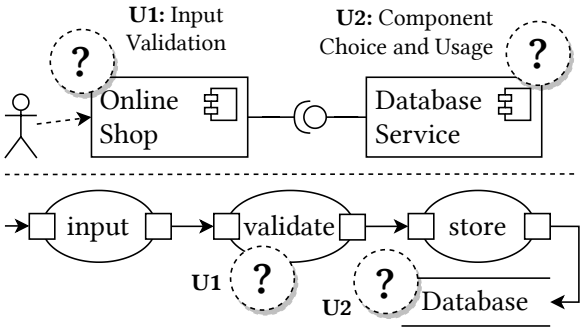

Figure 1. Component and data flow diagram of the running example

II. Architectural Modeling of Uncertainty

In this section, we discuss how to represent uncertainty sources and their impact on confidentiality. We focus on software-architectural uncertainty that can be represented and analyzed early in architectural abstraction [19]. For better understanding, we start with a running example that stresses the distinction between an uncertainty's source and its impact.

Figure 1 shows two diagrams of a simplified *Online Shop* [16] under uncertainty (**U1** and **U2**). The upper diagram shows a component diagram that represents one view of the software-architectural abstraction. It consists of an *Online Shop* component that is accessed by users and a *Database Service*. The lower diagram shows the extracted data flows [42] that represent the system's behavior from the viewpoint of the data [10]. Both diagrams are annotated with question marks that represent uncertainty sources, e.g., whether the input validation works as intended (**U1**) or which *Database Service* is in use (**U2**). Here, we see the difference between an uncertainty's source and its impact on confidentiality: If the input validation fails (**U1**), e.g., by forwarding confidential user details, the impact might become visible as late as the data flows to the *Database*. The locations of an uncertainty's source and its impact differ. Only considering the uncertainty's source location might thus hide the resulting problem.

Figure 2 summarizes this distinction between uncertainty sources and impacts on the data flow. An *Architecture Element* can be annotated with any number of *Uncertainty Sources*. Every source can have zero or more *Uncertainty Impact* locations within the software system. Regarding confidentiality, these are best represented using *Data Flow Diagram Elements* [19]. Here, the impact starts at the affected element and follows the data flow, as shown in the running example.

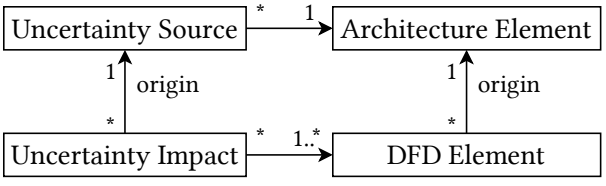

Figure 2. Relation of software-architectural uncertainty and its impact

In previous work, we investigated the different types of uncertainty that can affect confidentiality and defined a classification [19]. This classification describes uncertainty properties like location, manageability, or resolution time. Regarding confidentiality, we distinguish five types of software-architectural elements that can be affected by uncertainty: 1) *Component Uncertainty* is assignable to components in use that represent building blocks of a software system (e.g., Uncertainty **U2**). 2) *Actor Uncertainty* is assignable to actors in the system's environment like users, or hardware resources (e.g., unknown runtime conditions or deployment options). 3) *Behavior Uncertainty* is assignable to behavior descriptions like algorithms or user input (e.g., Uncertainty **U1**). 4) *Interface Uncertainty* is assignable to interfaces and their signatures that are used to connect system parts, i.e., component types (e.g., an undefined specification of transmitted data). 5) *Connector Uncertainty* is assignable to wires between components in use. This resembles interfaces but focuses on the communication rather than the type definitions (e.g., the interception of two components' communication). Our previous work indicates that these five types are sufficient to represent all sources of software-architectural uncertainty with an impact on confidentiality. We do not limit our modeling to a certain type of uncertainty, e.g., environmental uncertainty [5], or structural uncertainty [47].

While these types and their associated software-architectural elements are sufficient to document uncertainty sources, additional effort is required to also represent their impact. As discussed, confidentiality is best investigated using data flow diagrams [43]. We extend the unified modeling primitives of data flow diagrams by Seifermann et al. [42], which are also used in the running example and fit our uncertainty types.

Figure 3 shows our novel meta model that combines the data flow modeling primitives (highlighted gray) and the five uncertainty types. *Nodes* represent the system's structure and are affected by *Component Uncertainty* (e.g., Uncertainty **U2** impacts the *Database* node). *Flows* connect these nodes by transmitting data and are affected by *Connector Uncertainty*.

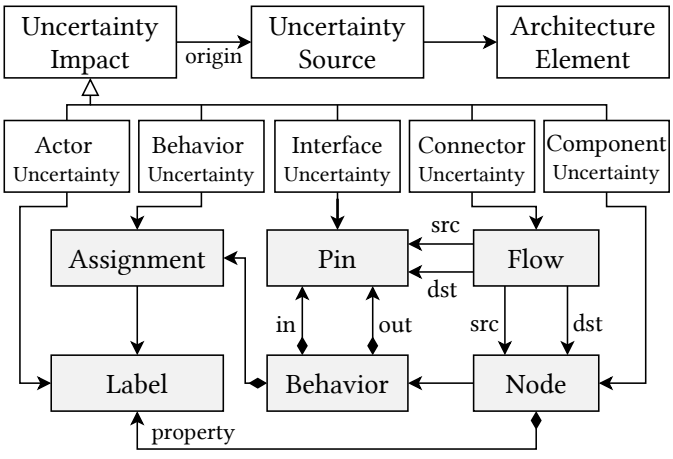

Figure 3. Meta model of data flow diagrams with uncertainty annotations

Every node has a behavior that consists of one or multiple *Assignments* of confidentiality-related labels, e.g., whether the input is validated (**U1**). These assignments are affected by *Behavior Uncertainty*. Nodes can also be described by *Labels* to represent their properties (e.g., their deployment location) and are affected by *Actor Uncertainty*. Last, *Pins* decouple flows from a node's behavior and function as interfaces that are affected by *Interface Uncertainty*. The *Behavior* has no associated uncertainty type because it only acts as a container for pins and assignments. We intentionally only model known uncertainty [19] and no quantitative information, e.g., the likelihood of possible scenarios. This reduces the knowledge required to make early predictions about uncertainty impacts.

In sum, this meta model enables us to connect the uncertainty types from the software architecture to their corresponding impact type. The representation as a data flow diagram simplifies the subsequent task of uncertainty propagation.

III. Uncertainty Impact Analysis

In this section, we exploit the previously presented modeling approach for analyzing the impact of uncertainty on confidentiality. We start with an illustrative description of the mapping and the propagation based on the running example. Afterward, we provide a mathematical description of the impact analysis and discuss the propagation algorithms in more detail.

Put simply, a data flow diagram consists of many connected nodes that represent the system's data processing. By mapping an uncertainty source from the architectural element to its counterparts in the data flow diagram, we can pinpoint the original impact locations. By following all outgoing data flows from these locations, we assess the set of all possible locations of confidentiality violations. In the running example, Uncertainty **U1** is annotated to the behavior of the *Online Shop* component and mapped to the *validate* node. Following all outgoing data flows, failure of the input validation can lead to violations in the *validate*, *store*, or *Database* node, e.g., by forwarding unfiltered user details. There is no propagation on the data flow *iff* the impact location already represents a data sink, e.g., the *Database* that is impacted by Uncertainty **U2**.

A. Formal Foundation of the Impact Analysis

A data flow diagram without cycles can be represented as a Directed Acyclic Graph (DAG) $G = (V, E)$ consisting of vertices V and edges E [9]. These represent nodes and data flows, respectively. Two vertices $u, v \in V$ are strictly partially ordered $u \prec v$ *iff* there exists a directed path from u to v, i.e., a (transitive) data flow. An induced subgraph $G[X]$ consists of a subset of vertices $X \subseteq V$ and their outgoing edges. In our running example, there exists a data flow from *validate* to *store*, thus $validate \prec store$. The data flow beginning from *validate* represents an induced subgraph of the full DAG.

To describe our analysis, we define, similar to Figure 2: Let $A = \{a_1, \ldots, a_n\}$ be the set of all architectural elements like components and interfaces. Let $S = \{s_1, \ldots, s_n\}$ be the set of all uncertainty sources and $I = \{i_1, \ldots, i_n\}$ the set of all uncertainty impacts. We name the annotation of an uncertainty source to an architectural element $a : S \to A$, e.g., $a(\textbf{\textit{U1}}) = OnlineShop$. We reuse the mapping [41] from an architectural element to its corresponding vertices of the data flow diagram as $m : A \to V$, e.g., $m(OnlineShop) = validate$.

The uncertainty impact analysis can be defined as function $u : S \to X \subseteq V$, where S represents all uncertainty sources. X induces a subgraph $G[X]$ of the data flow diagram, i.e., the part of the software system that is affected by the annotated uncertainty sources. The analysis consists of three steps: First, we propagate the uncertainty based on the structure of the software architecture by adapting the propagation rules defined by change impact analysis [36]. For example, altering an interface does affect both its caller and the callee. We define the structural propagation as $p_A : A \to A$. Second, we apply the previously defined mapping $m : A \to V$ from all affected architectural elements to their corresponding vertices of the data flow diagram. Third, we define the propagation along the data flow as $p_D : V \to X \subseteq V$. The previously mapped vertices represent the first affected nodes in the direction of the data flow, so that $\forall x \in X \subseteq V, \exists a \in A : m(a) \preceq x$. The induced subgraph $G[X]$ represents the full impact set including transitive effects. In sum, we define $u = p_D \circ m \circ p_A \circ a$.

In the running example, the annotated uncertainty sources **U1** and **U2** affect the *validate* and *Database* vertices after the structural propagation p_A and the mapping m. After following the data flow with p_D, both are part of the induced subgraph $G[X]$ with $\{validate, store, Database\} \in X$. In other words, a potentially erroneous input validation (**U1**) or an untrustworthy database service provider (**U2**) would cause confidentiality violations in these parts of the system. Note that this only represents a simplified scenario. Realistic data flow diagrams can consist of several hundreds of nodes [10].

B. Uncertainty Impact Propagation Algorithms

We realized this analysis approach based on the Palladio Component Model (PCM) [34] because of its widespread distribution and mature tool support. However, the realization is possible with any similar Architectural Description Language (ADL) or even a mixture of UML diagram types [40]. Our prototypical implementation is available online [18] to address the limitation of unavailable tooling of current research in this field [46]. To implement the propagation of uncertainty sources to their impact locations, we define propagation algorithms for each of the five uncertainty types discussed with Figure 3. In the following, we focus on the propagation of *Interface Uncertainty* that demonstrates the analysis comprehensively. The other algorithms are only described briefly due to space restrictions, but they are all implemented in our data set [18].

Algorithm 1 describes *Interface Uncertainty* propagation. Steps 2 – 4 represent the architecture-based propagation p_A: An uncertain interface has an impact on all components that implement the interface and also on all components that use such components. In the running example, an uncertain interface between the *Online Shop* and *Database Service* would affect both components. After identifying the PCM elements that represent this behavior (a component's *StartActions* and

Algorithm 1 Algorithm for Interface Uncertainty Propagation

Require: PCM software architecture model, corresponding data flow diagram, annotated interface uncertainty source

1: Find the interface annotated with interface uncertainty
2: Calculate the structural uncertainty propagation from the interface to all implementing components

for all components providing an implementation of the interface **do**
  for all method signatures of the interface **do**
    Find the start of the component's method implementation
    Add the *StartAction* to the list of impacts
  end for
end for

3: Calculate the structural uncertainty propagation from the interface to all calling components

for all components requiring an implementation of the interface **do**
  for all external calls of the component to other components **do**
    if the external call conforms to the interface **then**
      Add the *ExternalCallAction* to the list of impacts
    end if
  end for
end for

4: Calculate the structural uncertainty propagation from the interface to the system usage (similar to step 3)
5: Map all impacts to their corresponding data flow nodes
6: Calculate the uncertainty propagation along the data flows

for all mapped data flow diagram nodes **do**
  while the data flow diagram node is no file or data sink **do**
    Collect all successors of the node in the flow direction
  end while
end for

7: Calculate the impact set by finding the maximum discontiguous data flows, i.e., data flows not contained in each other

for all collected data flows **do**
  if a collected data flow represents a subgraph of this flow **then**
    Remove the subgraph from the collection of data flows
  end if
end for

ExternalCallActions), we map them to the extracted data flows [41]. Afterward, we propagate the uncertainty again along the data flows in Step 6 and combine the identified data flows in Step 7. This realizes the data flow-based propagation p_D and the construction of the induced subgraph $G[X]$.

Connector Uncertainty is similar to *Interface Uncertainty* but the uncertainty is propagated only for the annotated connection and not for all components that implement an interface. The propagation of the other uncertainty types consists of fewer steps because they already represent nodes rather than edges in the data flow diagram. *Behavior Uncertainty* is propagated to the corresponding data flow nodes, e.g., Uncertainty **U1** is propagated to the *validate* node. *Component Uncertainty* is propagated to the first nodes of all incoming data flows on the annotated component, e.g., Uncertainty **U2** is propagated to the *Database* node. Last, *Actor Uncertainty* is propagated to all nodes that describe the behavior of the actor, e.g., the *input* node describes the behavior of the user of the *Online Shop*. Steps 5 – 7 are similar for all uncertainty types.

In sum, this analysis enables us to propagate uncertainty in the software architectural model and along extracted data flows. The automated analysis yields an uncertainty impact set that predicts potential confidentiality violations.

IV. Case Study-based Evaluation

Our analysis predicts the impact of uncertainty on a software system's confidentiality by propagating uncertainty sources through the software architecture and along the data flows. Our goal is to minimize the overestimation of the potential uncertainty impacts while reducing the effort of both the modeling and the analysis. We validate the accuracy and effort reduction based on a real-world case study using the *Corona Warn App* [35]. In the following, we present our evaluation plan, design, and results and also discuss threats to validity.

A. Evaluation Goal, Questions, and Metrics

We use a *Goal Question Metric* plan [2] to evaluate the impact analysis. To enhance validity, this plan is close to the evaluation of architecture-based change impact analysis [36]. Our **Goal** is to evaluate the quality of the calculated impact set compared to manual confidentiality analysis. We use the common terminology [36] and call the *affected set*, the set of elements that are affected by uncertainty and would violate confidentiality in a manual confidentiality analysis. We consider the affected set to be the ideal result. Our automated analysis yields an *impact set*, i.e., the set of elements that are potentially affected. Here, the goal is that both sets are as close as possible with minimal overestimation [4]. To evaluate this goal, we ask the following questions:

- **Q1** How precise and complete is the calculated impact set compared to manual analysis?
- **Q2** Is there an effort reduction regarding the number of data flow diagram elements that software architects have to consider compared to manual analysis?

Question **Q1** evaluates the accuracy of our analysis [33]. An element of the impact set represents a *true positive (TP)* if it violates confidentiality and is thus also in the affected set. If the element is only in the impact set, it is a *false positive (FP)*. If an element of the affected set is not found by our analysis, it represents a *false negative (FN)*. We measure precision $P = \frac{TP}{TP+FP}$ (**M1.1**), recall $R = \frac{TP}{TP+FN}$ (**M1.2**), and the F_1 score, i.e., their harmonic mean, $F_1 = 2 \times \frac{P \times R}{P+R}$ (**M1.3**). For all metrics, *0* represents the worst, and *1* is the best possible value.

Question **Q2** evaluates the effort reduction [36]. We evaluate how many elements of the data flow diagram have to be considered by software architects while analyzing confidentiality. This evaluation is based on the data flow to enable a fair comparison to data flow-based confidentiality analysis [43]. We first consider the affected set that consists of correctly identified and all potentially non-identified confidentiality-violating elements of the data flow diagram. The ratio of the affected set to all elements *n* is calculated as $r_a = \frac{TP+FN}{n}$ (**M2.1**). The ratio of elements in the impact set, i.e., the number of all identified elements, is calculated $r_i = \frac{TP+FP}{n}$ (**M2.2**). Here, lower values are better.

B. Evaluation Data and Design

Our evaluation is based on a real-world case study using the *Corona Warn App* [35]. This represents a large enterprise system of systems that has been commissioned by the German government, developed by SAP and Deutsche Telekom during the COVID-19 pandemic, and downloaded more than 48 million times [35]. The contact tracing app exchanges keys between users via Bluetooth and handles highly sensitive data like COVID-19 test results. All source code and comprehensive design documentation are open-source and publicly available on GitHub [38]. Due to the strict confidentiality requirements and the rapid development in an uncertain environment, we consider this to be a suitable case study.

Based on the available documentation [38], we modeled the software architecture using Palladio [34]. Our model consists of 19 components, e.g., for processing and storing test results and exchanged keys, or user registration and authentication. We also modeled different deployment locations, usage scenarios, and the behavior of the essential functionality. The extracted data flows [42] consist of a total of 200 nodes. To evaluate all five uncertainty types, we define 4 uncertainty scenarios based on previous studies [3], [19]. Scenario **S1** collects uncertainty sources in one single component, e.g., the processing of confidential data. Scenario **S2** consists of environmental uncertainty, e.g., the deployment of servers. Scenario **S3** focuses on behavioral uncertainty, e.g., concerning the correct validation of user input. Scenario **S4** considers critical points in the system with a wide impact, e.g., due to a bug in the database. All models and details about the scenarios can be found in our data set [18].

We use our automated analysis to calculate the individual impact sets for all scenarios. These sets consist of data flow diagram elements that were identified based on the uncertainty propagation algorithms described in Section III. Afterward, we alter the modeled system to violate confidentiality for each annotated uncertainty source, e.g., by adding a bug in the validation of the user input or changing the server deployment locations. This represents the manual confidentiality analysis of what-if scenarios [40]. Manually modeling and analyzing confidentiality due to uncertainty requires more effort [21] but only yields actual violations, i.e., the affected set.

We count elements as *true positive (TP)* if they are contained in both sets and represent correctly identified confidentiality violations. Elements of the impact set that do not violate confidentiality and thus overestimate the affected set are counted as *false positive (FP)*. Each identified element of the impact set is either a *true positive* or a *false positive*. Data flow diagram elements that neither violate confidentiality nor have been identified by our analysis are classified as *true negative (TN)*. Elements of the affected set that were not identified by our analysis are counted as *false negative (FN)*.

C. Evaluation Results and Discussion

Table I shows the results of the evaluation. In all 4 scenarios, every confidentiality violation was predicted by the impact analysis, as expected. Thus, the affected set is always a subset of the impact set which results in the optimal recall R of *1.0*. This also implies that the ratio of the affected set r_a is a lower limit to the impact set ratio, i.e., $r_a \leq r_i \leq 1$.

Table I
EVALUATION RESULTS FOR ALL 4 UNCERTAINTY SCENARIOS

	S1	**S2**	**S3**	**S4**	**Average**
Precision P	0.838	1.000	0.840	0.882	0.890
Recall R	1.000	1.000	1.000	1.000	1.000
F_1 score	0.912	1.000	0.913	0.938	0.942
Ratio r_a	0.155	0.080	0.105	0.300	0.160
Ratio r_i	0.185	0.080	0.125	0.340	0.183

In Scenario **S1**, the uncertainty focuses on one single component and is primarily propagated in the data flow diagram rather than the architectural model. Because the confidentiality violation happens near to the data sink, the overestimation results in an F_1 score of *0.91*. The number of elements that would have been inspected is highly reduced with an impact set ratio r_i of *0.19*. In Scenario **S2**, the uncertainty is located in the environment of the software system. The confidentiality violations are also located in the data flow diagram nodes that represent the environment, e.g., in data sinks that represent databases. Thus, this scenario results in the optimal F_1 score of *1.0* and equal ratios $r_a = r_i$, because the impact set is equal to the affected set. Scenario **S3** focuses on uncertainty in the behavior of different parts of the system. Here, the uncertain validation of exchanged keys leads to confidentiality violations while inserting them into the database. This scenario yields an F_1 score of *0.91* and again highly reduces the required effort with $r_i = 0.13$. The last Scenario **S4** contains only wide-spreading uncertainty sources in central components like the main server database component. Although this results in a precision of *0.89*, still, all confidentiality violations are correctly predicted, and the required effort is reduced with an impact set ratio r_i of *0.34*. Despite the lacking precision, the analysis could still correctly exclude uncertainty impacts in 7 of the 14 extracted data flows, e.g., there is no impact on the test result servers if the application server fails.

Overall, the evaluation shows satisfying results. With a recall R of *1.0*, all confidentiality violations were correctly predicted without any false negatives. With an average precision of *0.89*, there is some overestimation which is common among impact analyses [3], [23], [36]. Here, we want to highlight the importance of optimizing for high recall (fewer false negatives) rather than high precision (fewer false positives). Impact analyses, like change impact analysis, are used to make early predictions of the possible outcomes of external influences. Especially regarding security-related properties like confidentiality, missing violating elements could have severe results. Thus, a certain degree of imprecision is acceptable as long as no relevant impact is overlooked, *and* the impact set is significantly smaller than the full system under study. In our evaluation results, this is the case as the number of elements that software architects have to consider is significantly reduced by 82% to a ratio r_i of *0.183* which is near the optimal

value of $r_a = 0.160$. Besides the number of elements, our impact analysis also reduces the modeling effort. To identify the confidentiality violations of the affected set, software architects have to additionally define data characteristics and data flow constraints [20]. Also, they must model possible effects of the uncertainty sources, e.g., by altering the system behavior or component wiring. With our approach, an element of the software architecture only has to be annotated with an uncertainty source without the need to define any what-if scenarios. The calculation of the impact set is fully automated.

D. Threats to Validity

We discuss threats to validity based on the guidelines by Runeson and Höst [37]. Regarding *internal validity*, a threat of model-based analyses is accurate modeling. Also, the modeled confidentiality violations can negatively impact the analysis results. To mitigate these issues, we used a case with comprehensive documentation of the software architecture that was already independently modeled in other work [3]. The *external validity* of our case study is limited by the system under consideration and by the comprehensiveness of uncertainty annotations. To that end, we built upon a domain-independent and already evaluated set of uncertainty types [19]. To maximize *construct validity*, we used a *Goal Question Metric* plan [2] that is closely oriented on the evaluation of change impact analysis [36]. Konersmann et al. [27] state a lack of replication packages and available tool-support. To address this and to increase *reliability*, we published a data set containing all evaluation data [18].

V. Related Work

We divide related work in three categories: Architecture-based uncertainty analysis, uncertainty-aware confidentiality analysis, and uncertainty propagation for self-adaptation.

Architecture-based uncertainty analysis: Research on uncertainty in software architecture uses modeling techniques to represent and analyze quality properties under uncertainty. Troya et al. [46] give an overview of the modeling of uncertainty, and Sobhy et al. [45] on evaluating software architecture while facing uncertainty. There are many approaches dealing with uncertainty in software architectures, e.g., by using fuzzy values [12], [29] or evolutionary optimization for design space exploration [28]. Such approaches can consider a wide range of quality properties such as performance, costs, or reliability. However, the coarse-grained modeling and analysis of confidentiality lacks in precise statements about confidentiality violations that help in identifying problems in the architecture.

Uncertainty-aware confidentiality analysis: To overcome this limitation of general-purpose architecture analysis, uncertainty-aware confidentiality analysis has been proposed. Walter et al. [47] extend PerOpteryx [28] with a data flow-based confidentiality analysis [43] to reason about structural uncertainty. Boltz et al. [5] reuse the same confidentiality analysis [43] to consider environmental uncertainty. In recent work, we used variability models to express uncertainty and analyze its impact on confidentiality [21]. However, all of these approaches require software architects to explicitly model the uncertainty impacts and do not consider their sources. As discussed, this requires more experience and effort and is also error-prone if the system or its environment changes.

Uncertainty propagation for self-adaptation: The propagation of uncertainty has also been proposed for designing models of cyber-physical systems [1]. Also at runtime, propagation can be used for monitoring and impact assessment as part of the MAPE-K loop [24]. The uncertainty types used for our meta model are also related to the modeling of self-adaptive systems [32]. A number of publications consider related uncertainty sources [13] and their management [6], [7], also at design time [50]. When considering multiple uncertainties to propagate, this is also related to the recently proposed uncertainty interaction problem [8]. We see much potential in integrating our propagation concept for a more precise uncertainty assessment by tracing the impacts of uncertainty on confidentiality. This is especially true for anti-fragile systems [15] that improve through uncertainty.

VI. Conclusion

In this paper, we presented an approach for modeling and analyzing the impact of uncertainty on confidentiality. Our meta model relates data flow diagram elements to different uncertainty types. The impact analysis propagates annotated uncertainty sources both in the software architectural model and in the data flow diagram. Our evaluation indicates that the calculated impact sets accurately predict potential confidentiality violations while reducing the required effort.

Propagating uncertainty helps software architects in handling uncertainty [1]. Architecture models can be annotated with uncertainty sources from existing collections [19] which helps in the documentation and to raise awareness. The analysis helps in predicting and mitigating confidentiality violations. Using a confidentiality analysis for this purpose would require software architects to understand and model the impact of uncertainty manually which requires more effort and expertise. The calculated models of our analysis can also be used for regression testing or to handle uncertainty at runtime [11].

In future work, we want to enhance both our modeling and analysis. By extending the expressiveness of our model and combining it with variability modeling [48], the precision of the predicted confidentiality violations shall be improved. Also, refined or extended propagation algorithms are imaginable. Last, we plan for additional evaluation domains, e.g., regarding cyber-physical systems or in the context of the increasingly connected mobility domain.

Acknowledgment

This work was supported by the German Research Foundation (DFG) under project number 432576552, HE8596/1-1 (FluidTrust), as well as by funding from the topic Engineering Secure Systems of the Helmholtz Association (HGF) and by KASTEL Security Research Labs. We like to thank Niko Benkler, who helped in developing the foundations of this work during his Master's thesis.

References

[1] M. Acosta *et al.*, "Uncertainty in coupled models of cyber-physical systems," in *MODELS-C*, 9, 2022, pp. 569–578.

[2] 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.

[3] N. Benkler, "Architecture-based uncertainty impact analysis for confidentiality," Master's Thesis, Karlsruhe Institute of Technology (KIT), 2022, 169 pp.

[4] S. Bohner, "Software change impacts-an evolving perspective," in *International Conference on Software Maintenance, 2002. Proceedings.*, ISSN: 1063-6773, 2002, pp. 263–272.

[5] N. Boltz *et al.*, "Handling environmental uncertainty in design time access control analysis," in *SEAA*, 2022.

[6] R. Calinescu, R. Mirandola, D. Perez-Palacin, and D. Weyns, "Understanding uncertainty in self-adaptive systems," in *2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)*, 2020, pp. 242–251.

[7] J. Camara, D. Garlan, W. G. Kang, W. Peng, and B. Schmerl, "Uncertainty in self-adaptive systems: Categories, management, and perspectives," Carnegie Mellon University, 1, 2017, Technical Report.

[8] J. Cámara *et al.*, "Addressing the uncertainty interaction problem in software-intensive systems: Challenges and desiderata," in *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems*, 23, 2022, pp. 24–30.

[9] G. Canfora, L. Sansone, and G. Visaggio, "Data flow diagrams: Reverse engineering production and animation," in *Proceedings Conference on Software Maintenance 1992*, 1992, pp. 366–375.

[10] T. DeMarco, "Structure analysis and system specification," in *Pioneers and Their Contributions to Software Engineering*, 1979, pp. 255–288.

[11] M. Derakhshanmanesh, J. Ebert, M. Grieger, and G. Engels, "Model-integrating development of software systems: A flexible component-based approach," *Software & Systems Modeling*, vol. 18, no. 4, pp. 2557–2586, 1, 2019.

[12] N. Esfahani, S. Malek, and K. Razavi, "GuideArch: Guiding the exploration of architectural solution space under uncertainty," in *2013 35th International Conference on Software Engineering (ICSE)*, ISSN: 1558-1225, 2013, pp. 43–52.

[13] N. Esfahani and S. Malek, "Uncertainty in self-adaptive software systems," in *Software Engineering for Self-Adaptive Systems II: International Seminar, Dagstuhl Castle, Germany, October 24-29, 2010 Revised Selected and Invited Papers*, 2013, pp. 214–238.

[14] D. Garlan, "Software engineering in an uncertain world," in *FoSER*, 2010, p. 125.

[15] V. Grassi, R. Mirandola, and D. Perez-Palacin, "Towards a conceptual characterization of antifragile systems," in *ICSA-C*, accepted, to appear, 2023.

[16] S. Hahner, "Architectural access control policy refinement and verification under uncertainty," in *ECSA-C*, 2021.

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

[18] S. Hahner, R. Heinrich, and R. Reussner, *Architecture-based uncertainty impact analysis to ensure confidentiality - data set*, 2023. DOI: 10.5281/zenodo.7753497.

[19] S. Hahner, S. Seifermann, R. Heinrich, and R. Reussner, "A classification of software-architectural uncertainty regarding confidentiality," in *ICETE*, to appear, 2023.

[20] S. Hahner *et al.*, "Modeling data flow constraints for design-time confidentiality analyses," in *ICSA-C*, 2021, pp. 15–21.

[21] S. Hahner *et al.*, "Model-based confidentiality analysis under uncertainty," in *ICSA-C*, accepted, to appear, 2023.

[22] R. Heinrich, K. Busch, and S. Koch, "A methodology for domain-spanning change impact analysis," in *SEAA*, 2018, pp. 326–330.

[23] R. Heinrich *et al.*, "Architecture-based change impact analysis in cross-disciplinary automated production systems," *JSS*, vol. 146, pp. 167–185, 2018.

[24] S. M. Hezavehi *et al.*, "Uncertainty in self-adaptive systems: A research community perspective," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 15, no. 4, 10:1–10:36, 2021.

[25] J. Isaak and M. J. Hanna, "User data privacy: Facebook, cambridge analytica, and privacy protection," *Computer*, vol. 51, no. 8, pp. 56–59, 2018.

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

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

[28] A. Koziolek, H. Koziolek, and R. Reussner, "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*, 20, 2011, pp. 33–42.

[29] I. Lytra and U. Zdun, "Supporting architectural decision making for systems-of-systems design under uncertainty," in *Proceedings of the First International Workshop on Software Engineering for Systems-of-Systems*, 2, 2013, pp. 43–46.

[30] S. McConnell, *Software project survival guide*. Microsoft Press, 1998.

[31] F. Oquendo, "Coping with uncertainty in systems-of-systems architecture modeling on the IoT with SosADL," in *SoSE*, 2019, pp. 131–136.

[32] D. Perez-Palacin and R. Mirandola, "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*, 2014, pp. 3–14.

[33] D. M. W. Powers, "Evaluation: From precision, recall and f-measure to ROC, informedness, markedness and correlation," *CoRR*, vol. abs/2010.16061, 2011.

[34] R. H. Reussner *et al.*, *Modeling and Simulating Software Architectures: The Palladio Approach*. The MIT Press, 2016.

[35] Robert Koch Institute. "Open-Source Project Corona-Warn-App." (2020), [Online]. Available: https://www.coronawarn.app/en/ (visited on 01/29/2023).

[36] K. Rostami, R. Heinrich, A. Busch, and R. Reussner, "Architecture-based change impact analysis in information systems and business processes," in *ICSA*, 2017, pp. 179–188.

[37] 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.

[38] SAP and Deutsche Telekom. "Corona-Warn-App." (2020), [Online]. Available: https : / / github . com / corona - warn - app (visited on 01/29/2023).

[39] P. Schaar, "Privacy by design," *Identity in the Information Society*, vol. 3, no. 2, pp. 267–274, 2010, Publisher: Springer.

[40] S. Seifermann, "Architectural Data Flow Analysis for Detecting Violations of Confidentiality Requirements," Dissertation, Karlsruhe Institute of Technology (KIT), 2022.

[41] S. Seifermann, R. Heinrich, and R. Reussner, "Data-driven software architecture for analyzing confidentiality," in *ICSA*, 2019, pp. 1–10.

[42] S. Seifermann, R. Heinrich, D. Werle, and R. Reussner, "A unified model to detect information flow and access control violations in software architectures," in *SECRYPT*, 2021, pp. 26–37.

[43] S. Seifermann, R. Heinrich, D. Werle, and R. Reussner, "Detecting violations of access control and information flow policies in data flow diagrams," *JSS*, vol. 184, p. 111 138, 1, 2022.

[44] A. Shostack, *Threat Modeling: Designing for Security*. John Wiley & Sons, 12, 2014.

[45] D. Sobhy, R. Bahsoon, L. Minku, and R. Kazman, "Evaluation of software architectures under uncertainty: A systematic literature review," *ACM Transactions on Software Engineering and Methodology*, p. 50, 2021.

[46] J. Troya, N. Moreno, M. F. Bertoa, and A. Vallecillo, "Uncertainty representation in software models: A survey," *Software and Systems Modeling*, 8, 2021.

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

[48] M. Walter *et al.*, "Architectural attack propagation in industry 4.0," *at - Automatisierungstechnik*, 2023, accepted, to appear.

[49] H. Weisbaum. "Trust in facebook has dropped by 66 percent since the cambridge analytica scandal." (2018), [Online]. Available: https: / / www . nbcnews . com / business / consumer / trust - facebook - has - dropped - 51 - percent - cambridge - analytica - scandal - n867011 (visited on 01/19/2023).

[50] J. Whittle, P. Sawyer, N. Bencomo, B. H. C. Cheng, and J. Bruel, "RELAX: Incorporating uncertainty into the specification of self-adaptive systems," in *2009 17th IEEE International Requirements Engineering Conference*, ISSN: 2332-6441, 2009, pp. 79–88.