S. Hahner, S. Seifermann, R. Heinrich, and R. Reussner, "A Classification of Software-Architectural Uncertainty Regarding Confidentiality," in E-Business and Telecommunications (ICETE), Springer, 2023. doi: 10.1007/978-3-031-36840-0_8

A Classification of Software-Architectural Uncertainty regarding Confidentiality

Sebastian Hahner, Stephan Seifermann, Robert Heinrich, and Ralf Reussner

KASTEL – Institute of Information Security and Dependability, Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany
{sebastian.hahner,robert.heinrich,ralf.reussner}@kit.edu, stephan.seifermann@alumni.kit.edu

Abstract. In our connected world, ensuring and demonstrating the confidentiality of exchanged data becomes increasingly critical for software systems. However, especially in early system design, uncertainty exists about the software architecture itself and the software's execution environment. This does not only impede early confidentiality analysis but can also cause data breaches due to the lack of awareness of the impact of uncertainty. Classifying uncertainty helps in understanding its impact and in choosing proper analysis and mitigation strategies. There already exist multiple taxonomies, e.g., from the domain of self-adaptive systems. However, they do not fit the abstraction of software architecture and do not focus on security-related quality properties like confidentiality.
To address this, we present a classification of architectural uncertainty regarding confidentiality. It enables precise statements about uncertain influences and their impact on confidentiality. It raises awareness of uncertainty properties, enables knowledge transfer to non-experts, and serves as a baseline for discussion. Also, it can be directly integrated into existing notions of data flow diagrams for uncertainty-aware confidentiality analysis. We evaluate the structural suitability, applicability, and purpose of the classification based on a real-world case study and a user study. The results show increased significance compared to existing taxonomies and raised awareness of the impact of uncertainty on confidentiality.

Keywords: Software Architecture · Uncertainty · Confidentiality

1 Introduction

Today's software systems become increasingly complex. With growing size and connections, ensuring software quality becomes a major challenge. This is especially true for security-related quality properties like confidentiality. Confidentiality demands that "information is not made available or disclosed to unauthorized individuals, entities, or processes" [17]. Violations of confidentiality cannot only harm user acceptance [45] but also have legal consequences [16]. To identify flaws early and to avoid costly repairs of running systems [4], design-time confidentiality analyses have been proposed [39,42]. Based on analyzing the software

architecture and its data against confidentiality requirements [15], the software design can be enhanced and statements about potential violations can be made. Here, data flow-oriented analyses became common because "problems tend to follow the data flow, not the control flow" [40].

However, especially in early development and in complex systems of systems, the software architecture is subject to uncertainty. Uncertainty describes "any departure from the unachievable ideal of complete determinism" [43]. This does not only affect decision making—also known as the *cone of uncertainty* [28]—but even blurs which decisions should be prioritized. When not managed properly, the lack of awareness of uncertainty can void a system's confidentiality. Also, the OWASP Top 10 [29] lists issues like *insecure design* as top security risks.

Multiple taxonomies were defined to better understand the nature of uncertainty [31,32,43]. However, they mostly originate from the domain of self-adaptive systems and do not focus on confidentiality. Consequences are the lack of applicability and an increase of ambiguity. The relation of software architecture, confidentiality and uncertainty remains unclear [13]. And while "there is growing consensus on the importance of uncertainty" [27], much is yet unknown regarding the impact of uncertainty on software systems [10]. Hezavehi et al. [27] conducted a survey on uncertainty. They find a "lack of systematic approaches for managing uncertainty" [27] and that uncertainty should already be addressed at design time. This statement is supported by the work of Troya et al. [41]. They conducted a Systematic Literature Review (SLR) and analyzed 123 papers. They state that software engineers require more help "to identify the types of uncertainty that can affect their application domains" [41].

In previous work [38], we presented a unified model to express data flows and analyze confidentiality violations in software architectures. We mined modeling primitives from existing approaches [37,42], defined a meta model and an analysis using label propagation. We also showed its integration into existing Architectural Description Languages (ADLs) like Palladio [33]. The underlying goal of a data-centered approach without predefined analysis goals or data flow constraints is the possibility of user-defined confidentiality analyses [39]. However, the previous approach was only able to analyze confidentiality with perfect knowledge, i.e., by excluding uncertainty about software systems and its data.

In this paper, we present a classification scheme of architectural uncertainty. This classification is specifically designed to express uncertainty on the architectural abstraction level regarding confidentiality. We build on the existing data flow model [38] and consider the impact of uncertainty on the software architecture's confidentiality. Here, we focus on *known unknowns*, i.e., uncertainty that can be identified but not always resolved immediately. The classification shall help architects to describe uncertain influences more precisely, find mitigation and analysis strategies, raise awareness of relevant uncertainty attributes, enable the reuse of knowledge, and serve as a baseline for discussion. We argue that making uncertainty explicit enhances the overall design and simplifies phase containment, i.e., fixing defects in the same phase as they appear. Here, uncertainty should not be avoided but become a source for improvement [11].

We start by discussing the state of the art in Section 2 and present the contributions of this paper (**C1** and **C2**) thereafter.

C1 First, we define *architectural uncertainty* and relate uncertainty to Architectural Design Decisions (ADDs) and confidentiality in Section 3. We extract relevant classification categories and discuss their applicability to describe the impact of uncertainty on confidentiality. Based on this, we present an uncertainty classification scheme in Section 4.

C2 Second, we show how the classification helps identifying the impact of uncertainty on confidentiality in Section 5. We demonstrate how the classified impact of uncertainty maps can be modeled using the unified modeling primitives [38]. We also provide a reference set of classified uncertainties.

The evaluation of the classification in Section 6 is based on the guidelines of Kaplan et al. [19]. The authors propose to evaluate the structure's suitability, the applicability, and the purpose. This enables us to make statements not only about the appropriateness and the quality of the classification as such, but also consider its reliability and ease of use, as required by *usable security* [36]. The evaluation also includes the metrics-based comparison to the state of the art, i.e., existing taxonomies of uncertainty. We conducted a user study with researchers from the software architecture domain and a real-world case study based on the German open-source contact-tracing app *Corona Warn App* [34].

The results show increased significance compared to existing taxonomies, i.e., better applicability and more precise classification. The user study shows that the classification scheme helps in understanding and analyzing uncertainty and is a satisfying base for discussions. This cannot only be seen in the gathered data but has also been independently reported by multiple study participants. Section 7 concludes this paper and gives an outlook on future application areas.

2 State of the Art

In this section, we give an overview of the state of the art based on three categories: Uncertainty taxonomies, uncertainty in software architecture, and ADDs.

Uncertainty taxonomies. To better understand uncertainty, researchers created several taxonomies [5,8,26,31,32,43]. Walker et al. [43] present a taxonomy of uncertainty using three dimensions. The *location* describes where the uncertainty can be found, e.g., in the model input or context. The *nature* distinguishes between epistemic (i.e., lack of knowledge) and aleatory (i.e., natural variability) uncertainty. Last, the *level* describes how much is known about the uncertain influence. Although this taxonomy has been the baseline for many others, it does not specifically aim to describe software-related uncertainty. Perez-Palacin and Mirandola [31] build upon this classification in the context of self-adaptive systems. They adjust the dimension *location* to better fit software models. Bures et al. [5] adapt this taxonomy again to "fit the needs of uncertainty in access

control" [5]. Although this work only considers access control in Industry 4.0 scenarios, it is also a good foundation for our classification. Esfahani and Malek [8] describe characteristics of uncertainty and hereby focus on the variability and reducibility of different sources of uncertainty. Mahdavi-Hezavehi et al. [26] propose a classification framework of uncertainty. They aim at architecture-based, self-adaptive system but do also not consider security, privacy, or confidentiality. Also related is the uncertainty template by Ramirez et al. [32]. They present a scheme to describe uncertainty sources for dynamically adaptive systems in requirements, design, and runtime. Due to the different scope, they describe uncertainty in software architecture as *inadequate design* which is not precise enough to identify the impact of architectural uncertainty on confidentiality.

Uncertainty in software architecture. Numerous approaches exist to handle uncertainty in software architecture [7,22,25]. For the sake of brevity, we only summarize to work related most. For an in-depth analysis, refer to the previously mentioned surveys [27,41]. GuideArch [7] and PerOpteryx [22] are approaches to explore the architectural solution space under uncertainty. Both approaches try to achieve optimal architectures under given constraints and degrees of freedom but do not aim at security-related properties like confidentiality. Lytra and Zdun [25] propose an approach to combine ADDs under uncertainty by utilizing fuzzy logic. Although this approach considers the software design, the representation of uncertainty as fuzzy values alone is not suitable to analyze confidentiality.

Architectural design decisions. The relation between design decisions and uncertainty has already been described more than two decades ago [28]. Kruchten [23] presents an ontology of ADDs. The author distinguishes between existence, property and executive decisions and provides an overview of ADDs attributes. This is especially relevant when considering uncertainty that can void existing decisions and require software architects to backtrack. Jansen and Bosch [18] see software architecture as a composition of ADDs. This shows how uncertainty, e.g., about the system context, can hinder good software design as the *best* decision might not be found. Although both approaches do not focus on uncertainty, they inspired our classification which is strongly coupled to architectural design.

3 Uncertainty, Confidentiality, and Software Architecture

In this section, we give an overview of uncertainty in software architecture. We propose the term *architectural uncertainty* and describe the relation of uncertainty, ADDs and confidentiality based on an exemplary architecture model. Afterwards, we discuss existing classifications of uncertainty and their applicability to describe the relation of architectural uncertainty and confidentiality.

When speaking about uncertainty in software architecture, we propose speaking in terms of *impact* rather than only considering the uncertainty's type or source. This enables software architects to focus on its mitigation during design, e.g., to enhance confidentiality. Also, when interpreting an architecture as set of ADDs [18], the impact is one of the most important properties to consider [28].

We understand *architectural uncertainty* as uncertainty, that can be described on architectural abstraction and where (early) awareness enables considering its impact on quality attributes like confidentiality. We do not use the term *design time*, as the real impact of the uncertainty might happen later, e.g., at runtime. We also do not only refer to *known unknowns*, as this only implicates awareness which is too imprecise. We refine this term by requiring the architectural abstraction, e.g., as part of an architectural model with specified impact on software-architectural elements, e.g., software components, interfaces, or hardware resources. Here, we also exclude higher orders of uncertainty [31] as their impact cannot immediately be expressed due to the lack of awareness. However, awareness can be raised with increasing knowledge, e.g., by asking a domain expert [30] or by using a classification scheme for systematic treatment [43].

Figure 1 shows an example of two architectural uncertainties and their impact on confidentiality [12]. The diagram represents a simplified *Online Shop* that consists of two components and two hardware resources. The first uncertainty **U1** is the allocation of the *Database Service* that stores the *Online Shop* data. The second uncertainty **U2** is the trustworthiness of the provider of the cloud service, a potential deployment location of the *Database Service*. Both uncertainties can be annotated in the architectural model and have a potential impact on confidentiality, e.g., due to legal requirements like the GDPR [6]. However, this example also shows the difference between awareness and mitigation: While deciding the allocation could resolve Uncertainty **U1**, Uncertainty **U2** potentially remains and requires further ADDs, e.g., the encryption of data.

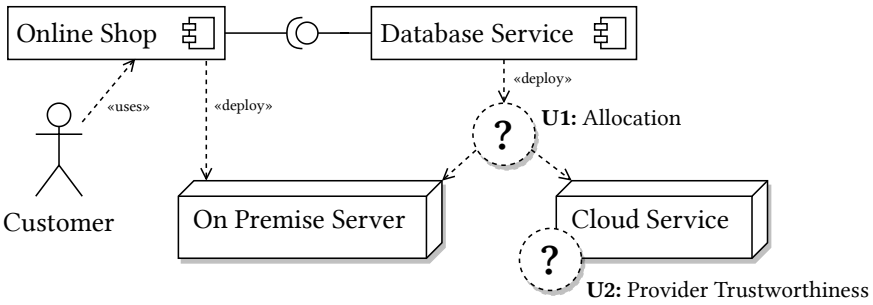

Fig. 1. Combined component and deployment architecture model with uncertainty

When dealing with architectural uncertainty, considering ADDs helps to structure the design process. In the beginning of this process, much is yet unknown or imprecise and ADDs are made under assumptions [28], e.g., that the provider is trustworthy in Uncertainty **U2**. Making this uncertainty explicit can help to mark decisions as challenged [23] and consider backtracking. While some uncertainty only exists due to not yet decided ADDs (e.g., Uncertainty **U1**), other cannot be reduced immediately [31] (e.g., Uncertainty **U2**). Still, creating aware-

ness of the impact of uncertainty can help refining the architecture and making more informed statements about confidentiality. This impact can be understood, modeled, simulated, analyzed, measured and—eventually—managed.

There are multiple relevant properties of ADDs that help in the mitigation of uncertainty. The number of solutions of related ADDs [18] can help to estimate whether the uncertainty can already be fully reduced at design time. We distinguish between closed sets that could at least partially be analyzed and open sets with a potentially infinite number of solutions or configurations. In our example, Uncertainty **U1** relates to the ADD of the allocation and represents a closed set. This can be analyzed, e.g., by combining design space exploration with dataflow analysis [44]. But even with a closed set of alternatives, one cannot guarantee that a given ADD might not be challenged in the future [23] due to changes in requirements or the system's execution context. Thus, when speaking about decisions under uncertainty, considering the probability, possibility and costs of revisions can help to quantify the risk. This awareness also helps in the prioritization of ADDs and deciding whether existing mitigation is sufficient.

However, only considering ADDs to understand the impact of uncertainty is not enough because uncertainty might not be directly connected to a single decision (e.g., resolving Uncertainty **U2**). Thus, we argue to also consider which architectural elements are affected, rather than only considering this transitively via the impact of ADDs. This does not only help understanding the consequences of uncertain influences but also helps to connect these to architecture-based analyses. In our example, modeling and analyzing the data flows [38,39] between the *Online Shop* component and the *Database Service* component helps determining whether the impact of Uncertainty **U2** is a problem for confidentiality.

To choose proper decision, mitigation, and analysis strategies, software architects need to be aware of the impact of uncertainty and be able to describe this impact precisely. To support this activity, we analyzed existing taxonomies and classifications of uncertainty. We gathered, assessed and adapted categories (i.e., dimensions, or characteristics) and options with the purpose of more precisely classifying uncertainty in relation to confidentiality. The following discussion is based on existing taxonomies [5,8,26,31,32,43] as well as recent systematic literature reviews and surveys on uncertainty [41,27]. Table 1 gives an overview of available categories and options derived from related work.

Uncertainty is often described with its *location*. However, there is no common distinction between the source and the impact of an uncertain influence. Also, there is no common understanding of the term *model* and its boundaries as the taxonomies originate from different research areas. Thus, uncertainty **U1** could be classified as *context*, *model structural*, *model technical*, and *belief uncertainty*. Such ambiguity can invalidate the purpose of a taxonomy. Regarding the *level*, two different approaches exist. Some classifications define the level based on the description of uncertainty, e.g., by using statistical means or scenarios. Others refer to the orders of uncertainty [1]. As already discussed, we focus on *known unknowns*, i.e., the first order of uncertainty. However, even in this order exist different nuances, e.g., recognized ignorance compared with statistical data.

Table 1. Available categories and options to classify uncertainty used in related work

Available Categories	Available Options
Location: Describes where uncertainty originates from or where it manifests itself within the system or model [5,26,31,43,41]	**Context**: system boundaries [31,43], user input [5], execution context [26]; **Model structural**: existence of elements [41], elements and their relationship [43,31], structural differences [26], components and their properties [5]; **Model technical**: software and hardware [43]; **Input**: input types [43], input values [31], measurement deviation [41]; **Parameters**: parameter calibration [43,31]; **System behavior**: actual behavior [5], including parameters and actions [41]; **Belief**: uncertain statements about system and environment [41];
Level: Describes how much is known about the uncertain influence and how the uncertainty can be described [43,26,31,5,1]	**Statistical**: Statistical data available [43,26]; **Scenario**: Possible scenarios available without statistical data [43,26,1]; **Recognized ignorance**: Awareness of uncertainty, but cannot be further described [43]; **Total ignorance**: Lack of awareness of uncertainty [43]; **Orders of Uncertainty**: No uncertainty (0th), known uncertainty (1st), lack of awareness, i.e., unknown unknowns (2nd), lack of awareness and process (3rd), meta-uncertainty (4th) [1,5,31];
Nature: Describes the essence and character of the uncertainty [5,26,31,43,41]	**Aleatory**: Uncertainty due to inherent variability or randomness [5,26,31,43,41]; **Epistemic**: Uncertainty due to a lack of knowledge [5,26,31,43,41];
Manageability: Describes whether the uncertainty can be reduced [8,43]	**Reducible**: Uncertainty can be fully reduced after acknowledgement [8,43]; **Irreducible**: Uncertainty cannot be further reduced at this point in time [8,43];
Emerging time: Describes at which state of the software development uncertainty arises [27,26,41,31,32]	**Requirements time**: As requirements are defined [32,41]; **Design time**: As the software system is designed [32,27,41,26]; **Verification**: At verification of software models [41]; **Testing**: During software testing [27,41]; **Implementation**: As the software gets implemented [41]; **Run time**: During software execution [27,26,41,31,32];
Impact on Quality: Describes how uncertainty affects quality properties [27]	**Performance**: Impact on performance [27]; **Resources**: Impact on resource consumption [27]; **Safety**: Impact on a system's safety [27];
Relationship: The relation between uncertainties [32]	**Directed**: Directed relationship or influences between uncertainties [32]; **Related**: Unspecified relationship between uncertainties [32];
Source: Potential sources of uncertainty [31,8,26,32]	Several publications list sources of uncertainty, e.g., human in the loop, abstraction, missing requirements, inadequate design, ...

Many classifications refer to the *nature* of uncertainty. However, this category is questioned [8,20] as it depends on the point of view and is often not clearly distinguishable. Thus, we prefer the category *manageability* since it focuses on the reducibility. Besides the category *emerging time*, we propose to also consider the resolution of uncertainty, i.e., at which time the impact of uncertainty can be understood and managed. Regarding design-time confidentiality analyses, it is valuable to know whether uncertainty can be directly analyzed (e.g., Uncertainty **U1**) or must be mitigated in later phases (e.g., Uncertainty **U2**).

The *impact on quality* is an important as not all uncertainties affect a system's confidentiality. To focus analysis capabilities, software architects must know about potential impacts and their severity. Although we focus on confidentiality, the analysis of multiple properties is possible, e.g., by using design space exploration combined with dataflow analysis, as demonstrated by Walter et al. [44]. The remaining categories are only defined for sources of uncertainty. Still, such information can be valuable, e.g., to describe the relation of uncertainty to ADDs like the allocation to Uncertainty **U1**.

4 Classification of Architectural Uncertainty

In this section, we present a classification scheme to aid software architects in understanding the impact of *architectural uncertainty* on confidentiality. This shall raise awareness of properties of uncertainty and their relevance for choosing appropriate ADDs and mitigation strategies. We intentionally speak of a classification rather than a taxonomy because we focus on a subset of uncertainty, i.e., known unknowns on architectural abstraction.

The classification scheme consists of 8 categories with a total of 27 options and is shown in Table 2. The categories are partially based on taxonomies of uncertainty [5,8,26,31,32,43], related work on ADDs [18,23] and ADLs like Palladio [33]. A category-based classification helps to group uncertainties and identify similar characteristics and mitigation approaches. Once classified, the information can be reused across different software architectures. This is possible due to the connection between architectural uncertainty and reusable ADDs. To create this classification, we assessed and adapted existing categories and combined or refined their options (see Section 3). We repeated this process until each category fulfilled its purpose, i.e., being able to describe and partition the impact of uncertainty on confidentiality in software architectures.

In this work, we focus on confidentiality requirements in the design time. Thus, the categories should be interpreted from an architectural viewpoint, e.g., while modeling a software system. In the following, we explain each category. We provide information about the rationale and possible benefits of applying each category. We also define whether the options are unordered (i.e., nominal) or ordered without defined distance (i.e., ordinal), and single or multiple choice. Last, we specify if the gained knowledge by classification can be reused, i.e., if it is specific for the uncertainty type, or the software architecture under investigation.

Location and Architectural Element Type. The first two categories are concerned with the location of the uncertainty impact. Previous taxonomies [5,26,31,43] already considered "where the uncertainty manifests itself within the model" [31] but did not explicitly relate to an ADL. Since the location of the impact is one of the most important properties for design-time confidentiality analysis, we connect this category with the viewpoints and element types of Palladio [33]. Compared to existing taxonomies, this enables more precise description and mitigation planning because we can model uncertainty and its relation to existing architecture elements. While *Location* is on the abstraction of the viewpoint (e.g., structure, or behavior), the *Architecture Element Type* describes the concrete elements (e.g., components, or hardware resources) affected by the uncertainty. Because this—especially regarding confidentiality—may affect multiple elements at once, both categories are multiple choice. Also, both categories are nominal as there is no order between location or element types. By understanding which elements and viewpoints are affected, software architects can assess responsibilities and evaluate mitigation methods. The knowledge gained about the location is uncertainty-specific and can thus be reused across architectural models.

Table 2. Categories and options of the architectural uncertainty classification.

Location: Describes where uncertainty manifests itself within the architecture	
System Structure	Structure, e.g., components, their static wiring, assembly, and allocation
System Behavior	Behavior of the system and its components as well as their communication
System Environment	System's context, including hardware resources and the external situation
System Input	Inputs provided by external actors, e.g., people using the software system
Architectural Element Type: Elements to which an uncertainty can be assigned	
Component	Assignable to software components, e.g., related to their allocation
Connector	Assignable to, e.g., wires between components, or communication
Interface	Assignable to interfaces, e.g., signatures, parameters, and return values
Hardware Resource	Assignable to hardware resources, e.g., servers, and external actors
Behavior Description	Assignable to behavior descriptions, e.g., algorithms or user input
Type: How much is known about the uncertainty and how can it be described	
Statistical Uncertainty	Uncertainty describable with statistical means, e.g., stochastic expressions
Scenario Uncertainty	Distinct scenarios depending on the uncertain outcome, no statistical means
Recognized Ignorance	Awareness of the uncertainty but no mitigation or description strategy
Manageability: Can more knowledge or appropriate means reduce the uncertainty	
Fully Reducible	Reducible, e.g., by acquiring more knowledge, or comprehensive simulation
Partial Reducible	At least partially reducible, e.g., by applying scenario-based mitigation
Irreducible	Uncertainty cannot be further reduced, e.g., due to its aleatory nature
Resolution Time: Time at which the uncertainty is expected to be fully resolved	
Requirements Time	As soon as requirements are defined, e.g., confidentiality requirements
Design Time	As soon as the systems is designed, e.g., its structure, or components
Realization Time	As soon as the system or parts of it are implemented and deployed
Runtime	As knowledge is gained from testing and system operations, e.g., monitoring
Reducible by ADD: Uncertainty resolvable by an architectural design decision	
Yes	Uncertainty can be reduced by taking an ADD, i.e., by designing the system in a way that the impact of the uncertainty is (partially) mitigated
No	Uncertainty is not resolvable or treatable by taking an ADD
Impact on Confidentiality: Potential impact on confidentiality requirements	
Direct	Direct impact on confidentiality, e.g., by directly affecting personal user data
Indirect	Impact only in conjunction with contextual factors, ADDs or uncertainties
None	No impact on confidentiality, e.g., if only publicly available data is affected
Severity of the Impact: Describes the severity if uncertainty is not mitigated	
High	Total loss of confidentiality, or sensitive data, e.g., an admin's password
Low	Access to restricted information could be obtained but the damage is limited
None	No loss of confidentiality expected at all

Type and Manageability. The next two categories specify how much is known about the uncertain influence and whether the uncertainty can be reduced. Other taxonomies [5,31] only specify this in terms of levels on a scale from knowledge to ignorance [1] which is too imprecise to classify uncertainty for mitigation. With the *Type*, we describe how much is known about the uncertainty, based on the definitions by Walker et al. [43]. All options represent *known unknowns* [31] but also specify how this knowledge can be represented for mitigation, e.g., if there is statistical data available. *Manageability* states whether we can control or reduce the impact of the uncertainty at design time (see *Data Protection by Design* [6]) or are only aware of it [8]. We do not consider the nature of the uncertainty [43] because the manageability is closer to the uncertainty's impact [8]. Both categories are ordinal and single choice. By understanding how much is known about an uncertain influence beyond awareness, one can choose appropriate mitigation methods or choose to gather specific knowledge. The classification depends on the context of the architecture under investigation. However, many uncertainty types tend to be categorized similarly across architectures. For instance, the allocation (see Uncertainty **U1** in Section 3) can usually be described *scenario-based* and be reduced in the design or realization time.

Resolution Time and Reducible by ADD. These categories relate uncertainty to the architectural design using ADDs. The *Resolution Time* is based on the phases of software development and can help to narrow down sources and responsibilities. Since we focus on the impact of uncertainty on confidentiality, we consider the expected full resolution time rather than the emerging time [27,31,32,41]. Also, we only include phases that are relevant from the point of view of design time analyses. The category *Reducible by ADD* specifies whether the impact of the classified uncertainty can at least be partially mitigated by a design decision. Making the connection between ADDs and uncertainty explicit [25] helps to prioritize, e.g., check whether multiple or critical uncertainty impacts can be tackled by a single decision. Both options are single choice, the options of the *Resolution Time* are ordinal, reducibility is considered to be nominal. Also, both categories are only uncertainty-specific and thus reusable.

Impact on Confidentiality and Severity of the Impact. The last two categories are used to quantify the impact of uncertainty on confidentiality requirements. To prioritize uncertainty with a critical impact, we combine the impact type with its severity. A direct *Impact on Confidentiality* can void confidentiality even without taking other factors, decisions, or uncertainties into account. Indirect impact relates to such contextual properties. *Severity of the Impact* is based on the confidentiality impact metrics of the open industry standard Common Vulnerability Scoring System (CVSS) [9]. They refer to a high impact if a total loss of confidential data or access to restricted information is expected. An impact that is rated low implies that data can be stolen, but the information could not be used directly or is limited. Both categories are single choice and ordinal. The knowledge gained by classification can help in clustering and prioritizing uncertainty and related ADDs but is specific for the architecture and its context.

5 Applying the Classification for Confidentiality Analysis

In this section, we apply the previously defined classification to analyze the impact of uncertainty on confidentiality. Confidentiality requirements are manifold. They include the legal restriction of data processing, e.g., personal data being regulated in the GDPR [6] but also organizational protection policies of restricted data like an administrator's password or encryption keys [9]. Thus, when speaking about the impact of uncertainty on confidentiality, one has to consider all relevant data in a software architecture. We achieve this by first classifying uncertainty and then considering the impact of this uncertainty in data flow-based confidentiality analysis [38,39].

Table 3. Exemplary classification of the uncertainty in the online shop example

	U1: Allocation	**U2**: Provider Trustworthiness
Location	System Structure	System Behavior / Environment
Architectural Element Type	Component	Behavior / Hardware Resource
Type	Scenario Uncertainty	Scenario Uncertainty
Manageability	Fully Reducible	Partial Reducible
Resolution Time	Realization Time	Runtime
Reducible by ADD	Yes	Yes
Impact on Confidentiality	Direct	Indirect
Severity of the Impact	High	Low

Table 3 shows an exemplary classification based on the *online shop* in Section 3. While the allocation (**U1**) represents uncertainty in the structure of the architecture and can be annotated to the deployable component, trustworthiness (**U2**) of a resource provider affects the behavior and is located in the environment of the system. Making the location explicit already helps in understanding the impact in terms of architecture models, e.g., describable as possible scenarios or model variations [30]. The allocation refers to the ADD of the same name and can thus be fully resolved at realization time, e.g., by limiting possible deployment locations. The trustworthiness remains unclear even at runtime, but can be partially mitigated, e.g., by enforcing encryption of data that flows to the *Database service*. Regarding legal confidentiality requirements [6], the allocation represents a serious and direct impact and can be prioritized over the trustworthiness which also depends on the allocation. Here, the classification helps to connect uncertainties and ADDs and also helps to prioritize. If a decision is revoked later, e.g., the deployment is changed to the *Cloud Service*, documenting classified uncertainties, assumptions and risk helps in reevaluation [23] and potential backtracking. Note, this classification is only exemplary and also allows to draw other conclusions due to the lack of contextual information in our simplified example. A comprehensive reference set of architectural uncertainties and their impact on confidentiality can be found in our data set [14].

Besides documentation and design, classifying uncertainty also helps in analysis and mitigation. Here, several approaches have been proposed (see Section 2). Perez-Palacin and Mirandola [30] distinguish between two mitigation paths: modifying the model (i.e., making the required ADD) and managing a model with uncertainty. These paths are a good fit to the previous discussion about manageability and reducibility. However, to choose one of these paths, software architects must be aware of the uncertain influence and require knowledge about its potential impact and related ADDs and elements of the software architecture.

To analyze confidentiality under uncertainty, we demonstrate how to manage a model with uncertainty and integrate the classification results into the previously presented data flow meta model [38]. This is possible because our classification has been designed for confidentiality and both—the classification scheme and the data flow meta model—relate to Palladio [33] as common ADL.

The unified modeling primitives [38] consist of *nodes*, *pins*, *flows*, *behavior* and *label assignments*. Nodes represent structural elements of software systems, e.g., processes, stores, or external entities. Pins represent their interfaces and flows are used to connect multiple nodes through their pins. Nodes have a defined behavior that assigns labels, e.g., based on labels at the node's input pins, constants, logical expressions or a combination of the above. We can follow the data flow by propagating the assigned labels through the system and applying the propagation function at each node. By comparing the labels at each node to defined requirements that are formulated as data flow constraints [15,39], we can analyze confidentiality.

Table 4. Mapping of architectural element types to data flow modeling primitives

Architectural Element	Data Flow Modeling Primitive	Impact of Uncertainty
Component	Node (process or store)	Existence and use of nodes
Connector	Flow	Existence of flows to nodes
Interface	Pin (input or output)	Existence and form of pins
Hardware Resource	Label assignment	Values of node assignments
Behavior Description	Behavior (label propagation)	Propagation function output

The modeling primitives are integrated into the control flow modeling of Palladio [33,38]. Table 4 shows the mapping of Palladio elements used in our classification to the meta model of the unified modeling primitives. It also shows how the impact of uncertainty on these elements can be transformed and represented in data flow models. As explained in Section 4, describing affected elements can be used for mitigation planning and analysis. Regarding the data flow modeling primitives, structural uncertainty can alter the existence of nodes, pins, and flows. Exemplary uncertain influences are component choices, interface definitions, or system configuration. Uncertainty can also arise from the context of a system, which is expressed by label assignments on nodes. Last, behavioral uncertainty could alter the output of affected label propagation functions.

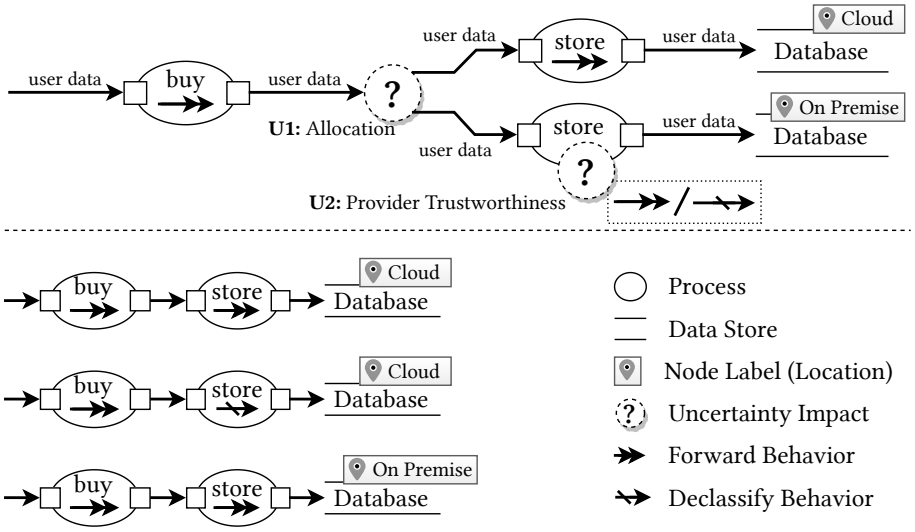

Fig. 2. Data flow diagrams of the running example with and without uncertainty

We demonstrate this mapping based on the *Online Shop* example whose uncertainties (**U1** and **U2**) have been classified in Table 3. Here, we are interested in the confidentiality of user data that is processed in the *Online Shop* component and then stored in a database. Figure 2 shows the resulting data flow diagrams, following the adapted notation presented in [38]. In the upper part, we show a non-deterministic data flow diagram under uncertainty (**U1** and **U2**). In the lower part, we show the impact of those uncertainties by rolling out the upper diagram and listing all resulting, deterministic diagrams. This is possible because both uncertainties represent *Scenario Uncertainty*, and we are also aware of all possible cases. Uncertainty **U1** is caused by an unknown allocation and represented by the two alternative label assignments *Cloud* and *On Promise*. Uncertainty **U2** is caused by the unknown provider trustworthiness that is represented by two alternative label propagation functions. While the *Forward behavior* passes incoming labels without modification, unauthorized behavior is represented as *Declassify Behavior* that changes propagated labels.

By defining constraints on these labels, we can analyze the resulting data flow diagrams on confidentiality violations. An exemplary constraint could prohibit unencrypted user data to be declassified. Although this is only the case in one of the possible data flows, to completely mitigate this issue, the *On Premise* allocation has to be chosen. Alternatively, user data can be encrypted. By altering propagation functions and data flow constraints, multiple scenarios under different uncertainties can be analyzed. This can be automated with tool support by combining the presented approach [37,38,39] with data flow constraints [15] and design space exploration. An initial approach has been realized by using PerOpteryx [22] to analyze confidentiality under structural uncertainty [44].

6 Evaluation

In this section, we present the evaluation of our classification. First, we define goals, questions and metrics and present the evaluation design. Afterwards, we discuss the evaluation results as well as threats to validity and known limitations.

6.1 Goals, Questions, and Metrics

Konersmann et al. [21] state the lack of guideline-based evaluation in current software engineering research. Especially taxonomies of uncertainty are often only evaluated based on application examples [5,31] or not evaluated at all [32,43]. To prevent this, we structure our evaluation according to the taxonomy evaluation method of Kaplan et al. [19]. The authors propose to use a *Goal-Question-Metrics*-plan [2] to evaluate the structure's suitability, the applicability, and the purpose of a classification. Table 5 summarizes the evaluation plan together with the evaluation results. In the following, we give an overview.

We evaluate the structure's suitability (**G1**), i.e., whether it permits the appropriate classification of objects under study by having the right scope and granularity. This includes the *generality* (**Q1.1**), where we measure if the classification is not too general but also not to specific. Here, low laconicity (**M1.1.1**) indicates a too fine-grained and low lucidity (**M1.1.2**) indicates a too coarse-grained classification. A good trade-off regarding the granularity is important because we want to be able to differentiate between uncertainties without assigning a separate class to every instance. Also, the granularity must fit to the purpose of classifying architectural uncertainty regarding confidentiality. The *appropriateness* (**Q1.2**) asks whether the classification is complete (**M1.2.1**), i.e., has enough categories, and whether it is sound (**M1.2.2**), i.e, has no unnecessary categories. On the one hand, we shall be able to classify every architectural uncertainty that can have an impact on confidentiality. On the other hand, categories and options that are never used, should not be maintained. Last, the *orthogonality* (**Q1.3**) evaluates whether the taxonomy has overlapping categories (**M1.3**). A lack in orthogonality implicates that options depend on each other and can be removed to increase preciseness. Overall, a classification with bad structural quality yield ambiguous results and shall be adapted.

We evaluate the applicability (**G2**), i.e., whether the classification is understandable and usable (see *Usable Security* [36]) by conducting a user study. Here, we consider the *reliability* (**Q2.1**), i.e., whether participants have consistent results (**M2.1**). An ambiguous classification with inconsistent results indicates a lack of preciseness. We evaluate the *correctness* (**Q2.2**) by comparing the classification results to a predefined gold standard and calculating the recall (**M2.2**) based on classification hits and misses. A lack in correctness indicates that users could not benefit from applying the classification. Last, we evaluate the *ease of use* (**Q2.3**) based on the System Usability Scale (SUS) [24] (**M2.3**). Additionally, we ask participants if they understand the categories and find them helpful and whether they experienced a knowledge gain by participating in the user study. A taxonomy has to yield consistent results to be usable.

We evaluate the purpose (**G3**), i.e., the classification's quality compared to existing taxonomies based on case study. We consider the *relevance* (**Q3.1**), i.e., whether each category helps the purpose of the classification (**M3.1**). In our case, the purpose is understanding the impact of uncertainty on confidentiality. The *novelty* (**Q3.2**) asks how many categories and options are new (**M3.2.1**) or adapted (**M3.2.2**). Here, the sum of both metrics indicates the strength of the relation of the classification to other taxonomies. Last, we consider the *significance* (**Q3.3**) by measuring the classification delta (**M3.3**) of our classification to other taxonomies. A positive delta indicates an increase in preciseness for which we aim. If the classification fails the evaluation of purpose, it represents no significant improvement over the state of the art.

6.2 Evaluation Design

Our validation comprises three properties. We evaluate the structural quality (**G1**) by analyzing the classification and we perform a user study to evaluate its applicability (**G2**). Our evaluation of purpose (**G3**) compared to the state of the art is based on the real-world case study of the *Corona Warn App* [34]. In the following, we describe the evaluation design in detail. Additional information such as raw evaluation data and questionnaires can be found in our data set [14].

Structural Evaluation. We start the evaluation by considering generality (**Q1.1**) and appropriateness (**Q1.2**). The calculation of the related metrics is based on the guidelines of Kaplan et al. [19]. The authors also provide tool support. For the metrics laconicity, lucidity, completeness, and soundness *1* represents the best and *0* the worst result. Besides the categories and options of our classification, this evaluation requires terms that describe uncertainties. We extracted a total of 38 terms of existing taxonomies (see Section 3). Examples are *Uncertainty fully reducible by acquiring enough knowledge* or *Uncertainty refers to user input*. The full list of terms can be found in our data set [14]. To evaluate the orthogonality (**Q1.3**) we construct a self-referencing orthogonality matrix based on our classification's categories and options. A category or option that is implied by another is not orthogonal and thus overlapping. Based on the 27 options of our classification, we evaluate all $27 * 27 - 27 = 702$ combinations.

User Study. We conduct a user study with ten researchers from the domain of software architecture. First, they complete a self-assessment, where they describe their prior knowledge related to the task, e.g., uncertainty, and software architecture. Then, we provide them with a one-page summary of our classification (cf. Section 4) with all categories and options and also an application example (cf. Section 5) that demonstrates how to use it. During the study, the participants have to classify two different tasks within 15 minutes time, respectively. Each task consists of an architecture diagram, a short description (cf. Section 3), and four uncertainty impacts to classify using our classification. We counterbalance the task order to mitigate learning effects and anonymize the participants' results. Last, they fill out a SUS (**Q2.3**) and a questionnaire related to their

understanding of our classification. No session takes longer than one hour to mitigate fatigue effects. After gathering all results, we measure the reliability (**Q2.1**) by calculating the percentage of agreement and the correctness (**Q2.2**) by comparing them to a predefined gold standard and calculating the recall.

Case Study. The *Corona Warn App* [34] is a German digital contact tracing app. It is publicly founded and open source. The source code of the app and the server, as well as comprehensive documentation can be found on GitHub[1]. This does not only include architecture documentation but also security analysis and risk assessment. By rolling back design decisions and considering solutions for problems and risks that are related to confidentiality, realistic uncertainties can be analyzed. We created a collection of 28 uncertainties that are possible during the design process based on the available documentation and ADDs. We use this collection as baseline for the evaluation of the purpose. For each category, we argue whether it helps to understand the impact of the uncertainties and is thus relevant (**Q3.1**). This extends the evaluation of generality (**Q1.1**) and appropriateness (**Q1.2**) based on our case study. We compare all categories to other taxonomies [5,8,26,31,32,43] of uncertainty to evaluate the classification's novelty (**Q3.2**). Here, we discuss the origin and adaption compared to the state of the art. We evaluate the significance (**Q3.3**) by classifying all 28 uncertainties with our classification and with other taxonomies [5,26,31] with a related purpose. As our goal is a higher precision for the impact of uncertainty, we aim for a positive classification delta, i.e., a higher number of uncertainty classes.

6.3 Evaluation Results and Discussion

In the following, we present and discuss the evaluation results for each question individually. Table 5 summarizes all goals, questions, and metrics as well as the evaluation results.

Table 5. Evaluation plan with goals, questions, metrics, and evaluation results

Goal	Questions	Metrics	Results
Structure's Suitability	Generality	Laconicity, Lucidity	0.95, 0.70
	Appropriateness	Completeness, Soundness	0.97, 1.00
	Orthogonality	Orthogonality Matrix	695 of 702
Applicability	Reliability	Inter-Annotator Agreement	0.69
	Correctness	Recall	0.73
	Ease of Use	Usability Score	68.25
Purpose	Relevance	Fraction of Relevant Classes	1.00
	Novelty	Innovation, Adaptation	0.49, 0.51
	Significance	Classification Delta	0.54

[1] https://github.com/corona-warn-app/

Structure's Suitability. To evaluate the structure's suitability (**G1**) of the classification, we consider generality (**Q1.1**) and appropriateness (**Q1.2**) and we measured laconicity, lucidity, completeness, and soundness. These metrics are defined for the leaves of a taxonomy, i.e., the 27 options of our classification. We gathered a collection of 38 terms R that describe the object under study, i.e., architectural uncertainties. In a laconic (**M1.1.1**) and thus non-redundant classification C, each term can be described using exactly one option. The *laconicity* is the fraction of terms that is uniquely describable: $laconicity(C, R) = \frac{36}{38} = 0.95$. We argue that the remaining two redundant terms are totally acceptable and originate due to the increased precision regarding confidentiality: User input can be classified as *Input* and *Environment* and uncertainty about non-confidentiality data has both *no impact* and *no severity*. In a lucid (**M1.1.2**) classification, each option describes no more than one term. *Lucidity* is the fraction of options that describe exactly one term: $lucidity(C, R) = \frac{19}{27} = 0.70$. Several terms are described by the same option, e.g., *Structure* describes both uncertainty in components and assembly. Another example is the *realization time* that includes implementation and deployment as this can be simplified from a design time perspective. Here, we decided that more fine-grained options would only harm the purpose of classifying and clustering uncertainties for understanding their impact and mitigation. In a complete (**M1.2.1**) classification, there is no term than cannot be described by at least one option. The completeness is thus calculated as fraction of terms that can be described: $completeness(C, R) = \frac{37}{38} = 0.97$. The completeness is reduced because we do not explicitly handle *known unknowns* that never resolve. From a design time point of view, it does not matter whether an uncertainty resolves at run time or never. In a sound (**M1.2.2**) classification, there are no unnecessary options that are not required to describe at least one term. $soundness(C, R) = \frac{27}{27} = 1.0$. The perfect result is expected, as we intentionally build the classification to fit our purpose. To evaluate the orthogonality (**Q1.3**), we constructed an orthogonality matrix (**M1.3**) and searched for implications between options. We found overlapping only in *7 of 702 cases*, e.g., uncertainty about the system's input implies a behavioral description and there exists the already discussed relation between *no impact* and *no severity*. However, none of the overlaps where comprehensive enough to justify the removal of a category or an option. All results were satisfying, so we continued with the second evaluation step as proposed by Kaplan et al. [19].

Applicability. To evaluate the applicability (**G2**), we conducted a user study with researchers with knowledge about software architecture but no or only little knowledge about uncertainty (for detailed results, please see our data set [14]). Ten participants classified two architecture models with four uncertainties each which yields a set of 80 classified uncertainties and 640 selected options [14]. To evaluate the reliability (**Q2.1**), we calculated the inner-annotator agreement (**M2.1**) by finding the largest consensus for each uncertainty and each category. The overall agreement is *69* percent. High agreement was measured in the categories *Location*, *Impact on Confidentiality*, *Severity*, and *Reducible by ADD*. The lowest agreement was measured in the category *Resolution Time*. One explana-

tion is the earlier description of the category, which was ambiguous and has thus been refined. To evaluate correctness (**Q2.2**), we compared the classifications to our gold standard and calculated a recall (**M2.2**) of *0.73*. Based on the participants' feedback, we find that the result can be explained with the short case descriptions of about the quarter of a page and the hard timing constraints. As also shown in Section 5, short descriptions of fictional architectures can leave a large room for interpretation. In view of the fact that the participants had no prior experience in classifying uncertainty, this result still is satisfying. Last, we evaluated the ease of use (**Q2.3**) with an average SUS (**M2.3**) score of *68.25*. In the questionnaire, most of the categories were considered understandable and helpful to describe the impact of uncertainty. The only outlier is the category *Type*. However, the value of this category has already been discussed in multiple publications [26,43]. Additionally, we demonstrated in Section 5 that this category helps in design-time confidentiality analysis. Also based on the participants' feedback, we summarize that our classification is a sufficiently useful tool to understand the impact of uncertainty but requires some familiarization. Most of the participants welcomed a lively debate about their classifications after the study sessions which is what we aimed for.

Purpose. To evaluate the purpose (**G3**), we first argue for the relevance (**Q3.1**) based on the fraction of relevant classes (**M3.1**). The purpose of our classification is to describe the impact of architectural uncertainty on confidentiality. The *Location* has already been discussed in other work [5,31]. We use *Architectural Element Type* because this enables the connection to architectural modeling and analysis. We demonstrated this in Section 5 with analyzing data flow models under uncertainty. We argue that *Type* and *Manageability* are better to describe uncertainty than only to refer to its level because this helps in choosing appropriate mitigation strategies. E.g., a scenario-based, reducible uncertainty can be handled different to a recognized, irreducible uncertainty. This has also been discussed in Section 5. *Resolution Time*, *Reducible by ADD* and both impact-related categories can be used in prioritization together with connected ADDs. This prioritization and connection to ADDs is important because it helps structuring the software design and also helps focusing modeling and analysis capabilities. We close that no category can omitted without significantly reducing the expressiveness and thus the fraction of relevant classes is *1.0*. For the novelty (**Q3.2**), we counted new (**M3.2.1**) and adapted (**M3.2.2**) categories and options. A category or option is adapted if it is adopted or derived from another classification. Examples for adopted categories are the *Resolution Time* or *Severity of the Impact*. Examples for new options are the *partial reducibility*. We find the hard distinction between manageable and irreducible not precise enough for design time mitigation. Also in our running example, we were able to better understand and partially reduce the impact of uncertainty. A full discussion of all 8 categories and 27 options can also be found in our data set [14]. We are right in the middle between innovation ($\frac{17}{35} = 0.49$) and adaption ($\frac{18}{35} = 0.51$). This is expected as we build upon existing taxonomies but extended them to fit our purpose. Last, we evaluate the significance (**Q3.3**) by calculating the classifica-

tion delta (**M3.3**). Our classification is able to distinguish the 28 uncertainties of the case study into 21 classes. Other taxonomies yield between 4 and 8 classes. Thus, the classification delta is $\frac{21-8}{28} = 0.54$. As we aimed for higher precision, a value higher than 0 is sufficient. We conclude that the metrics indicate that our classification fits its purpose, also compared to the state of the art.

6.4 Threats to Validity

We discuss threats to validity based on the guidelines by Runeson and Höst [35]. Regarding *internal validity*, the biggest threat is the evaluation of structure's suitability that has only been performed by the authors and is thus based on their limited experience. However, we adhered to the metrics and guidelines of Kaplan et al. [19]. The *external validity* and generalizability of our results is threatened by the number of participants in our user study and the selection of the case study. Still, we argue that both were large enough to identify a general trend of applicability and purpose. The participating researchers deal with software architecture in their daily work and the *Corona Warn App* is a large open-source system that is actively observed by the community. Additional information on the construction of the case study can be found here [3]. To face threats to *construct validity*, we applied a GQM-based evaluation plan [19]. Additionally, the SUS provides a standardized format, which might not fit the evaluation of classifications. We mitigated this using a questionnaire which yields similar results regarding the usability. To enhance *reliability* and replicability by other researchers, we publish all evaluation data [14].

6.5 Limitations

We are aware of three limitations of our classification. First, we only focus on confidentiality as central quality attribute. While this reduces the applicability, we did this intentionally to obtain more precise results for mitigation. For example, the focus on confidentiality at design time enables the connection to a data flow meta model [38] for design time confidentiality analysis. Second, the classification is focused on Component Based Software Engineering (CBSE) and works best based on architectural modeling. This was also an explicit design decision due to fit existing modeling [33,38] and analysis [37,39,42] approaches for confidentiality. Still, most categories are general enough to be used even without explicit models, e.g., *Type*, *Manageability*, or *Resolution Time*.

Last, the classification provides no assistance for the transitive impact of uncertainty. The direct impact is often not the location where the uncertainty affects confidentiality and where it can be mitigated. In our exemplary application in Section 5, the uncertain provider's trustworthiness could not only directly affect the data base but indirectly other parts of the system. However, to face such propagation effects, a precise description of uncertainty—such as our classification—is required in the first place. Additionally, the propagation, mapping, and analysis of uncertainty for design-time confidentiality analysis requires tool-support as "detecting confidentiality issues manually is not feasible" [37].

7 Conclusion

In this paper, we presented a classification of architectural uncertainty to describe its impact on confidentiality. We explained the relation of software architecture, uncertainty, and confidentiality based on existing classifications. Then, we defined our uncertainty classification and showed how the gained knowledge can be used for mitigation. We demonstrated the mapping of classified uncertainties to an existing data flow model for design time confidentiality analysis. This shall help software architects to better understand the different uncertainty types and analyze their impact. The evaluation showed satisfying results regarding the structural quality, the applicability, and the significance of our classification compared to the state of the art.

Our work benefits software architects in terms of more precise statements about architectural uncertainty and awareness of its different types. As several categories are reusable, this also enables knowledge transfer and reduces the required expertise for mitigating uncertainty. It is also a good baseline for discussion and assessment of uncertainty impacts. This has also been confirmed by our user study. Our classification helps to understand uncertainties and also to document and to prioritize ADDs. This can shorten the span of required backtracking in case of challenged decisions. By making design time confidentiality analyses uncertainty-aware, more comprehensive statements about confidentiality are possible. This shall also help in building more resilient software systems.

In future work, we want to tackle the limitation of manual annotation and analysis of the impact of uncertainty. Based on the presented classification, we want to create assistance for modeling and propagating uncertainty through the software architecture. We also want to create guidelines for better understanding different types of uncertainty and their potential impact on confidentiality. This shall not only enhance reliability and correctness of uncertainty classification, but also enable software architects to identify and mitigate the transitive impact of uncertainty and to make statements about confidentiality under uncertainty.

Acknowledgments. 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 (46.23.03) of the Helmholtz Association (HGF) and by KASTEL Security Research Labs. We like to thank Niko Benkler, who helped in developing this classification during his Master's thesis. We also like to thank all participants of the user study.

References

1. Armour, P.G.: The five orders of ignorance. Communications of the ACM **43**(10) (2000)
2. Basili, V.R., Weiss, D.M.: A Methodology for Collecting Valid Software Engineering Data. TSE pp. 728–738 (1984). https://doi.org/10.1109/TSE.1984.5010301
3. Benkler, N.: Architecture-based Uncertainty Impact Analysis for Confidentiality. Master's thesis, Karlsruhe Institute of Technology (KIT) (2022)

4. Boehm, B., Basili, V.: Defect reduction top 10 list. Computer **34**(1), 135–137 (2001)
5. Bures, T., et al.: Capturing Dynamicity and Uncertainty in Security and Trust via Situational Patterns. In: ISoLA. pp. 295–310. Springer (2020). https://doi.org/10.1007/978-3-030-61470-6_18
6. Council of European Union: REGULATION (EU) 2016/679 (General Data Protection Regulation) (2016), https://eur-lex.europa.eu/eli/reg/2016/679/2016-05-04, accessed 05/11/2022
7. Esfahani, N., et al.: GuideArch. In: ICSE. pp. 43–52 (2013). https://doi.org/10.1109/ICSE.2013.6606550
8. Esfahani, N., Malek, S.: Uncertainty in Self-Adaptive Software Systems. In: Software Engineering for Self-Adaptive Systems II, pp. 214–238. Springer (2013). https://doi.org/10.1007/978-3-642-35813-5_9
9. FIRST: CVSS v3.1 specification document, https://www.first.org/cvss/v3.1/specification-document#2-3-Impact-Metrics, accessed 05/11/2022
10. Garlan, D.: Software engineering in an uncertain world. In: Proceedings of the FSE/SDP workshop on Future of software engineering research - FoSER '10. p. 125. ACM Press (2010). https://doi.org/10.1145/1882362.1882389
11. Grassi, V., Mirandola, R.: The Tao way to anti-fragile software architectures: the case of mobile applications. In: ICSA-C. pp. 86–89. IEEE (2021). https://doi.org/10.1109/ICSA-C52384.2021.00021
12. Hahner, S.: Architectural access control policy refinement and verification under uncertainty. In: ECSA-C (2021)
13. Hahner, S.: Dealing with Uncertainty in Architectural Confidentiality Analysis. In: Proceedings of the Software Engineering 2021 Satellite Events. pp. 1–6. GI (2021)
14. Hahner, S., et al.: Companion data set, https://doi.org/10.5281/zenodo.6814107
15. Hahner, S., et al.: Modeling Data Flow Constraints for Design-Time Confidentiality Analyses. In: ICSA-C. pp. 15–21. IEEE (2021). https://doi.org/10.1109/ICSA-C52384.2021.00009
16. Isaak, J., Hanna, M.J.: User Data Privacy. Computer **51**(8), 56–59 (2018). https://doi.org/10.1109/MC.2018.3191268
17. ISO: ISO/IEC 27000:2018(E) Information technology – Security techniques – Information security management systems – Overview and vocabulary. Standard (2018)
18. Jansen, A., Bosch, J.: Software Architecture as a Set of Architectural Design Decisions. In: WICSA. pp. 109–120 (2005). https://doi.org/10.1109/WICSA.2005.61
19. Kaplan, A., et al.: Introducing an evaluation method for taxonomies. In: EASE. ACM (2022). https://doi.org/10.5445/IR/1000145968, accepted, to appear
20. Kiureghian, A.D., Ditlevsen, O.: Aleatory or epistemic? does it matter? Structural Safety **31**, 105–112 (2009). https://doi.org/10.1016/j.strusafe.2008.06.020
21. Konersmann, M., et al.: Evaluation methods and replicability of software architecture research objects. In: ICSA. IEEE (2022), accepted, to appear
22. Koziolek, A., et al.: PerOpteryx: automated application of tactics in multi-objective software architecture optimization. In: QoSA-ISARCS. pp. 33–42. ACM (2011). https://doi.org/10.1145/2000259.2000267
23. Kruchten, P.: An Ontology of Architectural Design Decisions in Software-Intensive Systems. In: 2nd Groningen workshop on software variability. pp. 54–61 (2004)
24. Lewis, J.R.: The system usability scale: past, present, and future. International Journal of Human–Computer Interaction **34**(7), 577–590 (2018). https://doi.org/10.1080/10447318.2018.1455307
25. Lytra, I., Zdun, U.: Supporting architectural decision making for systems-of-systems design under uncertainty. In: SESoS. pp. 43–46. ACM (2013). https://doi.org/10.1145/2489850.2489859

26. Mahdavi-Hezavehi, S., et al.: 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). https://doi.org/10.1016/B978-0-12-802855-1.00003-4
27. Mahdavi-Hezavehi, S., et al.: Uncertainty in Self-Adaptive Systems: A Research Community Perspective. ACM TAAS (2021)
28. McConnell, S.: Software project survival guide. Microsoft Press, Redmond, Wash. (1998)
29. OWASP Foundation: Owasp top 10:2021 (2021), https://owasp.org/Top10/, accessed 05/11/2022
30. Perez-Palacin, D., Mirandola, R.: Dealing with uncertainties in the performance modelling of software systems. In: QoSA. pp. 33–42. ACM (2014). https://doi.org/10.1145/2602576.2602582
31. Perez-Palacin, D., Mirandola, R.: Uncertainties in the modeling of self-adaptive systems. In: ICPE. pp. 3–14. ACM (2014). https://doi.org/10.1145/2568088.2568095
32. Ramirez, A.J., et al.: A taxonomy of uncertainty for dynamically adaptive systems. In: SEAMS. pp. 99–108 (2012). https://doi.org/10.1109/SEAMS.2012.6224396
33. Reussner, R.H., other: Modeling and Simulating Software Architectures: The Palladio Approach. The MIT Press (2016)
34. Robert Koch Institute: Open-Source Project Corona-Warn-App (2020), https://www.coronawarn.app/en/, accessed 05/11/2022
35. Runeson, P., Höst, M.: Guidelines for conducting and reporting case study research in software engineering. Empirical software engineering **14**, 131 (2009). https://doi.org/10.1007/s10664-008-9102-8
36. Sasse, M.A., Flechais, I.: Usable security: Why do we need it? how do we get it? O'Reilly (2005)
37. Seifermann, S., Heinrich, R., Reussner, R.: Data-driven software architecture for analyzing confidentiality. In: ICSA. p. 1–10. IEEE (2019). https://doi.org/10.1109/ICSA.2019.00009
38. Seifermann, S., Heinrich, R., Werle, D., Reussner, R.: A unified model to detect information flow and access control violations in software architectures:. In: SECRYPT. p. 26–37. SCITEPRESS (2021). https://doi.org/10.5220/0010515300260037
39. Seifermann, S., et al.: Detecting violations of access control and information flow policies in data flow diagrams. JSS (2022). https://doi.org/10.1016/j.jss.2021.111138
40. Shostack, A.: Threat Modeling: Designing for Security. John Wiley & Sons (2014)
41. Troya, J., et al.: Uncertainty representation in software models: a survey. Software and Systems Modeling (2021). https://doi.org/10.1007/s10270-020-00842-1
42. Tuma, K., et al.: Flaws in Flows. In: ICSA. pp. 191–200. IEEE (2019). https://doi.org/10.1109/ICSA.2019.00028
43. Walker, W.E., et al.: Defining uncertainty: a conceptual basis for uncertainty management in model-based decision support. Integrated assessment **4**(1), 5–17 (2003). https://doi.org/10.1076/iaij.4.1.5.16466
44. Walter, M., et al.: Architectural optimization for confidentiality under structural uncertainty. In: ECSA'21 Post-Proceedings. Springer (2022), accepted, to appear
45. Weisbaum, H.: Trust in facebook has dropped by 66 percent since the cambridge analytica scandal (2018), https://www.nbcnews.com/business/consumer/trust-facebook-has-dropped-51-percent-cambridge-analytica-scandal-n867011, accessed 05/11/2022