What do you assume? A Theory of Security-Related Assumptions

Sophie Corallo*, Thomas Weber*, Lars König*, Leonie Schmidt
Frederik Reiche*, Jan Keim*, Tobias Hey*, Anne Koziolek*
*Karlsruhe Institute of Technology
Karlsruhe, Germany
forename.lastname@kit.edu

***Abstract*—Assumptions play a significant role in software engineering. Implicit, inconsistent, or invalid assumptions about the system can have a high impact on the system, especially on its security. Even though there are several approaches for managing assumptions in security engineering, most of them are highly specific to their domain and phase in software development. For holistic assumption management, a general understanding of security-related assumptions and their relation to other artifacts is required. Founded on a Grounded Theory-based approach, including nine interviews with security researchers and a literature review of 53 scientific publications on assumptions, we observe common properties of security-related assumptions. Based on that, we propose a definition and several properties of security-related assumptions. With two surveys and 148 participants from software engineering, we validate the definition of security-related assumptions and general assumptions in software development. We also confirm the close relationship between assumptions and requirements. Lastly, our study underlines the importance of documenting assumptions and their features.**

I. Introduction

Assumptions play a significant role in various aspects of software engineering, as evident from prior research [1, 2, 3]. Throughout software development and evolution, assumptions are made concerning, for example, requirements, design, and system performance. The complexity of modern systems, often characterized by extensive dependencies and component reuse, necessitates numerous assumptions. Regrettably, many of these assumptions are impromptu and left implicit. Consequently, conflicts or mismatches between assumptions can lead to inconsistencies or invalidation of assumptions. The repercussions of implicit, inconsistent, and invalid assumptions can be profound, causing requirement violations, suboptimal designs, miscommunication, customer dissatisfaction, increased costs, and many system issues, including security vulnerabilities [3]. It is crucial to explicitly manage and address these assumptions to prevent adverse consequences and enhance the overall robustness of software systems.

Many of these problems can be prevented by assumption management (AM). In this paper, we define *assumption management* for software systems as the systematic development and maintenance of explicit assumptions in software systems. This covers identifying, describing, evaluating, maintaining, tracing, monitoring, reusing, and organizing assumptions [3]. Every aspect is important: Without identifying assumptions, they cannot be made explicit. Descriptions of assumptions should be comprehensive for all stakeholders to support the communication between them. Maintaining, tracing, and monitoring assumptions is important for identifying inconsistent or mismatching assumptions and offering an early and continuous feedback loop. Evaluating assumptions helps to manage potential vulnerabilities and risks. Finally, the reuse and organization of assumptions provide an overview of the weaknesses of a design or system. Thereby, AM enables the evaluation of potential vulnerabilities and the estimation of associated risks similar to requirements engineering.

In systems with elevated security standards, explicit assumption management becomes critical. Failures in such systems, such as those used in cars, medical systems, finance, or power supplies, can, among others, severely affect user privacy or safety. Security engineering faces numerous challenges, with many issues arising from undocumented and implicit assumptions that give rise to vulnerabilities [4]. Consequently, assumptions hold paramount importance in the context of security, and the proper handling and documentation of these assumptions through AM can significantly enhance the robustness and resilience of security-critical systems. Several techniques seek to analyze and ensure the security of systems, such as formal analyses, threat models, and security by design. However, every security analysis depends on assumptions. A prominent example of a severe security failure, despite the use of multiple security mechanisms and security analyses, is the Equifax data breach in 2017. There, multiple unjustified assumptions were made [5]. For example, there was no defense-in-depth applied to the attacked Automated Consumer Interview System (ACIS): A single vulnerability of the Apache Struts web framework could be used to gain access to the system. Thus, the web framework was mistakenly assumed to be trustworthy. Additionally, the security teams assumed that the used vulnerability scanners guaranteed the identification of vulnerabilities, but in fact, the applied scanners have known "blind spots" [5]. To the best of our knowledge, there is neither a holistic approach to AM in security-related systems that can help uncover such (a combination of) unjustified assumptions nor a generally applicable notion that helps to do so.

Even though there are several established terms for assumptions, e.g., context assumptions, trust assumptions or architectural assumptions, there are diverse understandings [3]. Moreover, the notion *assumption* seems to occur mainly

in the early phases of software development and to fade out during software maintenance and evolution. Furthermore, assumptions seem to be subjective, context-dependent, and dynamic regarding time and software engineering (SE) phases. Thus, a common understanding of *assumptions* is required for effective assumption communication and management. So far, most approaches cover only a few aspects of AM and restrict themselves to a specific type of *assumptions* [3].

Consequently, our guiding research questions are

- RQ1 What is a security-related assumption, and how is it linked to other software development concepts?
- RQ2 Is there a difference between security-related and general assumptions in software development?
- RQ3 What information should be included when assumptions are documented?

To answer these questions, this paper aims to provide a definition for security-related assumptions for AM. The proposed definition is derived by a Constructivist Grounded Theory-based approach on structured interviews with security experts from academia and a literature review. The definition is extended to general assumptions in software development and validated with a survey. The results of the survey also emphasize the importance of the documentation of assumptions and their features and highlights their relation to requirements.

We first describe related works in Section II and our research method in Section III. Section IV presents the theory-building process and the resulting definition. We then answer our research questions in Section V, before we discuss threats to validity in Section VI and conclude the paper in Section VII.

II. Background & Related Work

Assumptions can be found in various areas of SE [2, 3] under a variety of notions: Parnas defines interfaces as assumptions between modules [6], Garlan et al. describe assumptions as expectations of components to their environment [7], and Lehman and Fernandez-Ramil use them to describe the assertions of a system to its context [8]. This section highlights different notions of assumptions and describes existing models.

Even though *assumption* is widely used, it is often not well-defined [3]. In a systematic mapping study on assumptions and their management, Yang et al. investigate the notion of assumptions in SE [3]. Even though assumptions have diverse notions and usages, they find some similarities: Assumptions have uncertainty, have a dynamic nature, and are subjective and context-dependent. In sum, Yang et al. identify seven types of assumptions across twelve definitions [3]. The most frequent types are *context*, *trust*, *architectural*, and *early architectural*. *Context assumptions*, also called *environmental assumptions*, are broadly defined but usually target the context as described by Jackson [3, 9]. These assumptions concern system externals, like the behavior of another system, a user, or a resource. *Trust assumptions* are "choices, statements, or opinions that concern the behavior and properties of the system" [3]. *Architectural assumptions* are "implicit design decisions as well as the rationale and context behind these decisions" [3]. In contrast, *early architectural assumptions* target building blocks of envisioned architectures before any design decision is made. Although this taxonomy helps to understand different kinds of assumptions and to get an overview, transitions are fluent.

In security, *trust assumption* is widely used [10, 11, 12]. Viega et al. describe them as trust relationships between entities, i.e., software components or companies [10]. Erroneous trust assumptions often stem from incomplete requirements and miscommunication. Viega et al. conclude that trust must be considered throughout the software engineering process. Therefore, trust relationships are directly connected to system requirements and design decisions. Tracing such relationships also allows developers to check whether the final implementation places too much trust in involved entities, e.g., end users.

Like Viega et al., Haley et al. observe the direct impact of trust assumptions on the realization of a system, the satisfaction of its requirements, and its security [11]. Even though they evolve different understandings of *trust assumption* over years of research, they mainly refer to context assumptions. They describe each assumption with a narrative, a domain, an effect, preconditions, and justification and link it to the respective requirement [13]. Given these properties, they analyze if the system requirements are fulfilled in the assumed context.

ISO/IEC 15408-1 for criteria of IT security [14] addresses context assumptions and defines them similarly to Yang et al. [3]. They are used to create security objectives and derive requirements and specifications. Thereby, assumptions are treated similarly to threats and organizational policies.

Threats depend on assumptions but can also be derived from them [15]. Assumptions can also be stated to ease the application of threat models or other attacker-centric analyses, e.g., to assume attacker abilities, system, and context to analyze timing attacks on software systems [16]. In a study on 122 threat models, Van Landuyt and Joosen showed on 845 assumptions that they are used to exclude potential threats, often undocumented, and about the system under analysis [17]. Moreover, 74% of the assumptions concern functionality or architecture of the system. Van Landuyt and Joosen investigate the use, role, function, coupling to the system, and the specificity of assumptions in threat models further [18]. Assumptions help with threat modeling by limiting the scope. If these threat models are then used for requirements elicitation, requirements depend on assumptions [19, 20].

There are few explicit models for assumptions. Lago and van Vliet distinguish between environmental, organizational, and managerial assumptions to document assumptions of a product family explicitly [21]. From their observations of examples, they derive a model for assumptions. In the model, an assumption belongs to exactly one category and can be refined into other assumptions. They reflect a layering of assumptions: An assumption can impact features and their structural elements. Vice versa, features can realize and fulfill assumptions.

Tirumala provides a formal language for documenting environmental assumptions in a machine-checkable architecture analysis language format [22]. Thus, assumptions are stated from components to other components or systems to ensure the correct functionality of the components. These assumptions

have several aspects, such as a validity time frame, a criticality, and a compositional scope. The time frame divides assumptions into three classes: static assumptions, which do not change at all; system configuration assumptions, which do not change during execution; and dynamic assumptions, which may change during execution of the system. Criticality classifies assumptions into critical and non-critical. Whereas invalid critical assumptions compromise core functionalities and lead to a failure of a component, violated non-critical assumptions degrade or compromise non-essential functionality. Finally, the compositional scope differentiates between public assumptions, which need to be satisfied by system externals, and private assumptions, which can be validated system-internally.

In contrast, Mamun et al. propose a formalization for architectural assumptions [23]. Moreover, they allow cross-cutting assumptions, which relate to more than two components. They manually classify, annotate, and capture assertions in *Alloy assertions* to represent and check them. However, the model does not explicitly allow refining or linking assumptions.

Another model for architectural assumptions is derived from two industrial case studies by Yang et al. [24]. Here, a stakeholder always provides an architectural assumption. It can be bidirectionally related to software artifacts, namely requirements, architectural design decisions, and system components, and can lead to risks. As attributes, assumptions have a name, description, state, and rationale. Assumptions change over time and, therefore, have a history. Similar to other models, architectural assumptions can be linked.

Lewis et al. divide assumptions in control, environmental, data, usage, and convention assumptions to represent them in XML [25]. However, the representation most often only provides the type and description. Based on the assumption types of Lewis et al., Ordibehesht proposes an approach to explicate architectural assumptions [26]. In their model, an assumption requires at least a name and a description. To link assumptions to architectural elements, users should define dependencies that mark the impact of assumptions on components and which components realize the assumption. Additionally, both the dependency and the general assumption have custom fields that can be filled with additional helpful information, e.g., the type of assumption.

Conclusion In this section, we have shown that there are several notions of *assumption* in SE. Based on Yang et al. [3], we sorted the assumptions into three classes: context, trust, and architectural assumptions. Even though there is a wide landscape of assumption definitions, we can confirm and extend the commonalities between different assumptions mentioned by Yang et al.. As assumptions are mainly made explicit for the communication, an understandable description of their content is mandatory. That there is a relation between requirements and assumptions is also highlighted by several authors [13, 14, 24]. This relation is strengthened by the connection to threats [15, 18, 16, 17]. Further relations emerge when solving the dependency between trust assumptions and architectural assumptions. Thereby, system-related assumptions always relate to design decisions and, thus, to components, their implementation, or other artifacts [3, 24, 26, 23, 21].

The classification from Yang et al. and existing notions of *assumption* have the problem that assumptions are only defined for specific contexts and software development phases. However, assumptions have a dynamic nature and flexible usage. This requires to define and refine assumptions regarding contexts, software development phases, and viewed artifacts. To the best of our knowledge, there is no conceptual model that addresses this problem.

III. Research Method

In this section, we describe our theory-building approach. Section III-A introduces Constructivist Grounded Theory and our process. We describe the interview design, participant acquisition, and interview procedure in Section III-B. Lastly, we explain the data selection and coding procedure for the literature review in Section III-C1.

A. Constructivist Grounded Theory

Grounded Theory (GT) originates from social sciences and has been applied to various domains, including software engineering [27, 28]. The main goal of GT is to inductively generate theory from data by immediate and continuous data analysis. This includes using analytical codes, writing memos, constantly comparing new data and the created theory, sorting memos, and formulating a theory. GT values open mindsets. Thus, prior extensive literature reviews should be avoided, even though everything can be considered as data in GT. Usually, GT is based on unstructured texts, like interviews, but can also use structured artifacts, like diagrams or quantitative data. When new data does not change a theory, theoretical saturation is reached and the theory is considered well-supported.

There are several types of GT [28]. In this paper, Charmaz's Constructivist Grounded Theory (CGT) is used as it allows researchers to begin with a research question [29]. Compared to others, CGT also permits a limited literature review to fit the purpose of the study. The evaluation of CGT faces four criteria: saturation, originality, soundness, and usefulness.

1) Rationale & Data: Beforehand, we observed different notions of *security-related assumption* that caused misunderstandings. As literature lacks uniform definitions and models, we chose CGT to derive a concept and definition. We first interviewed security experts to retrieve an open-minded theory. For a more saturated theory, we additionally included 53 publications on assumption notions and assumption management as data.

2) Roles: We distributed the theory-building to several people: Two contributors conducted the prior literature review to get an overview of the topic ([13, 21, 25, 17, 3, 1, 30, 31, 32, 33]). Based on this, we formulated the interview questions. One contributor conducted the interviews and started the interview-based theory-building with another contributor. Afterward, four contributors discussed the memos regarding the literature review and thus derived the final concept and theory.

B. Interviews

The interviews aimed to capture the concept of *security-related assumption* from security researchers. We chose security researchers because of their domain expertise and involvement in security analyses and development of mechanisms, and the alignment with our objective to identify a relevant conceptual model. To maintain the integrity of their notion during the interview, we refrained from providing a clear definition of *security-related assumption* or *security requirement*. The interview design followed several guidelines [34, 35].

1) Interview Design: We designed a structured interview pattern with developing question groups as recommended by Knott et al. [36]. Thereby, we structured our interviews in three phases: An initial phase with low complexity, a main phase with the core questions, and an end phase that can include questions that require reflection. Each phase consists of three to four questions. The interview design can be found in our replication package [37].

The first question (Q1) asks the participants about their research topics. This vitalizes the participants and allows the interviewer to grasp the context and domain of the interviewee. Q2 then shows and describes an exemplary scenario. The scenario describes the Dutch smart grid by Van Aubel and Poll and is later used as a common case [38]. To make the participants familiar with the scenario, they were asked to relate their research topic to the case (Q3).

In contrast to the initial phase, the main phase focuses on security-related assumptions and requirements the participant experiences in their work. Thus, Q4 asks for assumptions and requirements the participant makes in their research. The interviewer further asked for the addressed targets, e.g., users, contexts, or models, as well as assumptions or requirements that could not be ensured. Subsequently, the interviewee was asked to generalize their assumptions and requirements to other contexts (Q5). Like Q5, the sixth question also refers to Q4 and asks for the impact of the invalid assumptions or requirements (Q6). Finally, Q7 asks about the relationship between assumptions and requirements.

We designed the last phase to slow down the interview. Q8 asks the participants for their opinion on the documentation of assumptions. Similarly, Q9 asks whether the participants explicitly considered assumptions before the interview. Finally, the participants should state whether they see benefits in thinking about assumptions and their explicit definition.

2) Participant Acquisition: We acquired participants by sending a pre-survey to 50 security experts from a German university. However, of the sixteen respondents only nine agreed with automated transcriptions. The interviewees were doctoral and postdoctoral researchers concerned with different facets of security. Their interests ranged from theoretical computer science, e.g., encryption, to more practical aspects of security, e.g., uncertainty in software development.

3) Interview Procedure: Each interview was planned for 40 minutes and conducted in German. To ensure comparability across different interviews, we used fixed time intervals for the question groups and used the same interviewer. The pilot study with two interviews confirmed the comprehensibility of the questions and did not uncover sequence effects [39].

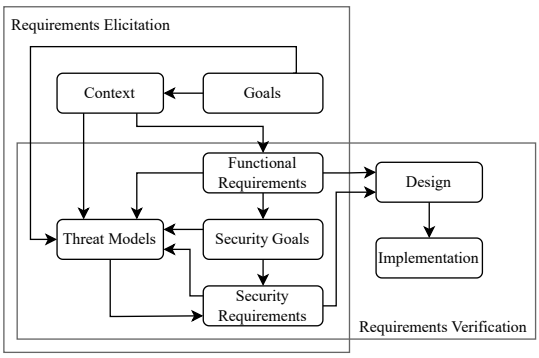

Figure 1. Artifacts in requirements engineering

At the start of each interview, we obtained permission to record the interview and its later transcription. The participants signed a data privacy declaration, and gave their consent for recording, transcription, pseudonymized storage, and use of their data for research, conforming to the General Data Protection Regulation. However, one of the nine interviewees did not agree to publish their pseudonymized interview transcripts. Except for these, all artifacts of the interviews can be found in our replication package [37]. For every interview, we logged the date, time, location, and potential interruptions in an interview protocol *(excluded from the replication package due to the doubly anonymous submission)*.

C. Literature Review

As proposed by Charmaz, we avoided an extensive prior literature review [29] before the interviews. However, to increase the theoretical saturation, we afterward conducted an extensive literature review with 53 papers. We selected literature as described in Section III-C1 and sorted it by requirements engineering activities described in Section III-C2. Then, we coded the literature per activity as described in Section IV-B. Between coding steps, we refined our memos.

During the first sightings of the literature, we observed that many approaches in assumption management are specific to their context and bring their own notion and concept of assumptions. This is also the case for approaches that connect two artifacts. As the goals of assumption management activities seems to be highly related to the goals of requirements engineering activities, we sorted papers respectively to their addressed requirements engineering activities.

1) Data Selection: There are several studies for assumption notions and management in software development and evolution [40, 4, 24, 3, 2]. We reused the referred literature from these and extended it with papers we found on our own. As the studies are usually not related to security, we especially searched for works in this domain.

2) Requirements Engineering Activities: To identify different activities in requirements engineering, we assume a similar SE process as Haley et al. [41, 42, 43]. As shown in Figure 1, the

process starts with the definition of business goals from which the context is derived. Based on the context, the functional requirements are elicited. Non-functional requirements, except for security, are neglected in the figure. This leads to security goals that help to elicit security requirements. In parallel, the threat model can use existing information to generate misuse cases and potential threats. These help to limit the scope. The security requirements can be used to update threat models or system context. Else, security and functional requirements are used to design the system. Lastly, the design is implemented. The framework can be divided into two intertwined parts: requirements elicitation and requirement verification.

IV. Theory Building Process

This section presents the intermediate steps of the theory-building via our CGT-based approach. Section IV-A describes the three initial memos that were created based on the interviews. Section IV-B explains how the third memo evolved due to the literature review. All related artifacts and the referenced interviews can be found in the replication package [37].

A. Interview Results

In parallel to the interviews, the conductors of the interview-based theory-building started to code, memo, and analyze the data together. For that, they used the tool QualCoder [44]. They retrieved a first version of the memo, i.e., a diagram sketch, after three sessions, a second version after two additional interviews, and a third version after four additional interviews. Due to limited space, only the resulting concept is described. However, a more detailed description of the first memos can be found in our replication package [37].

In the first interviews (*I4754, I2681, I8563*) the participants came from the domains: usability of security, cryptography, and confidentiality. The interviewees mentioned diverse assumptions, e.g., on the trust of users in other parties, their technical understanding of the system under investigation (SuI), attacks and attacker models, the design, and the implementation. We observed assumptions about the context of the SuI that are predetermined by the environment and their fulfillment cannot be influenced. For example, "the existence of trustworthy entities" (*I2681*). In contrast to that, participants mentioned also controllable assumptions about the SuI and its fulfillment, e.g., "everything [that is] described in the model will be implemented in the end" (*I8563*). Furthermore, two interviewees mentioned the consequences of the invalidation of their assumptions on the successful development or usage of the SuI. Finally, all interviewees mentioned the relationship between requirements and assumptions, whereas some stated that "requirements or assumptions, [..] are two different things" (*I2681*). They suggested that in contrast to requirements, *guarantee*s and assumptions are taken for granted, e.g., mathematical constraints like $P \neq NP$.

The next two interviews covered the topics of security by design and consistency preservation among confidential artifacts (*I3827, I0367*). These raised our focus on the criticality of an assumption. This criticality seemed to relate to the consequences of the invalidity of an assumption. Moreover,

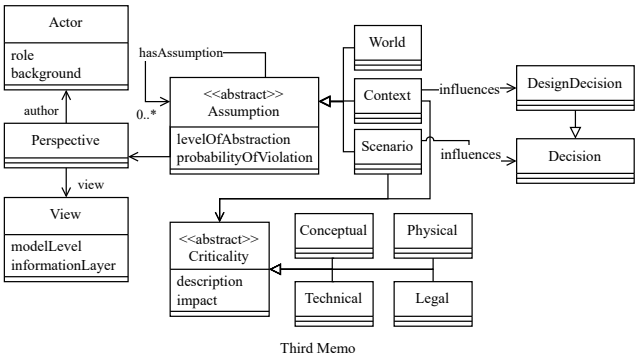

Figure 2. Concept evolved from the interviews

a more detailed separation of the targets of assumptions was needed, indicated by statements like "for some application scenario [it] is not part of our work [..], because we are [..] one abstraction level over it" (*I0367*). This also lead to first thoughts that assumptions could be refined. We further observed that the importance of an assumption correlates with its probability of violation and a high level of abstraction as it may affect bigger parts of the SuI. However, not every invalidated assumption seems to have the same effect on the SuI. Up to this point, we observed influences on physical properties, e.g., sufficient energy supply, technical features, e.g., the correct implementation of a used library, legal regulations, e.g., alignment with the GDPR, and conceptual characteristics, e.g., "there are no other connections [between components] beyond [those in the design]".

The last interview-based coding was influenced by four additional interviews (*I8615, I2608, I5266, I1614*) and some literature [45, 1]. Similar to Zave and Jackson, we observed a differentiation between world, context, and scenario assumptions. In contrast to the other kinds, world assumptions like "we assume, that no meteor will hit our office", cannot be influenced and were thus not directly related to the criticality of the system. We also observed a differentiation of assumptions via their influences: Whereas scenario assumptions seem to influence general system-related decisions, context assumptions seem to have a direct influence on design decisions. Finally, we saw that there are different ways to look at the same assumption. To describe these perspectives, the actor, their role, their background, and their view on the system seem to be important. The view should conclude all information the actor sees of the system. This includes the model level, e.g., and architecture model or source code, as well as an information layer that describes what kind of annotations are included in this model, e.g., annotations for data flow analyses.

Figure 2 shows the final concept derived from the interviews.

B. Literature Review Results

In this section, we describe the coding of the literature review based on the selected publication (c.f. Section III-C1). We also consider the notions presented in Section II. We

classified the work according to requirements engineering activities (c.f. Section III-C2) and coded it per activity.

1) Requirements Elicitation: For requirements elicitation, it is most important to gain a representative overview of the system. Therefore, AM is useful to support software engineers in making assumptions, describing them, and finding conflicts.

Goals: The first assumptions can be elicited during the refinement of goals. Broadnax et al. describe that the selection of a security mechanism to operationalize a security-related goal (which is a kind of design decision made to fulfill this goal) implies assumptions about the security of the mechanism [46]. For example, consider the "goal *the data must be exchanged confidentially*. The black box mechanism *virtual private network (VPN) protocol* fulfills the [..] goal under the separating assumption *security of encryption schemes* [..] and *correctness of the VPN protocol implementation*" [46]. Some of these assumptions can be verified (e.g., cryptographic proof of the encryption scheme), but the others remain "trust anchors". Similarly, Wang et al. use a graph-like structure to relate assumptions with requirements and goals [47].

Coding: The descriptions of the relations between assumptions, requirements, and design decisions align with our observations from interviews. Remarkably, Broadnax et al. do not distinguish between requirements and assumptions [46].

Context: The connection between goals and requirements is often impacted by the context. This also applies to assumptions [48, 49]. In their systematic literature review on non-technical assumptions in solution designs, Bazarhanova and Smolander observe that designs can depend on human-user assumptions [48]. Therefore, some works refine assumptions on the context: Faily and Fléchais refine assumptions on user populations to support the design of secure systems [50]. Thus, they explicitly model users and add them to the model. Assumptions on the users are described with Toulmin's Argumentation Model [51]. Toulmin's model consists of a claim, grounds, warrant, backing, modal qualifier, and rebuttal. The claim describes the actual assumption. Grounds, warrant, and backing cover the argument for the justification of the proposition. The modal qualifier is about the degree of certainty of the argument for the claim. The rebuttal contains statements that challenge the validity of the claim.

Coding: The works emphasize and confirm the influence of context assumption on design decisions. Faily and Fléchais use a description for their assumption including explicit justification arguments that was not yet covered by our concept [50]. Finally, the works confirm the self-relation of assumptions.

Security Requirements: For security requirements, threat models are often used to elicit requirements or to check them for completeness. El-Hadary and El-Kassas elicit security requirements based on abuse frames and model threats using a catalog [20]. As the threats depend on assumptions, they have to be refined. Only if the analyst is not able to detail the assumptions further, a trust assumption remains. This process is similar to the goal and requirement refinement [47]. However, in contrast to other works [47, 46], El-Hadary and El-Kassas do not see assumptions as requirements and design decisions as their solution, but make assumptions on the decisions. They state: "[We] might need some design assumptions in order to identify if there exists any vulnerability" [20].

Coding: Unlike previous works, El-Hadary and El-Kassas introduce a different relation between design decisions [20]. As a result, we adopted two relations between assumptions and design decisions in our concept: one for assumptions that represent requirements and are solved by design decisions, and another for design decisions that are based on assumptions.

Threat Models: Several studies show that most assumptions in threat modeling are used to exclude threats [17, 18]. Many assumptions are also relevant outside of security and highly coupled to the system [17, 18]. Van Landuyt and Joosen state that many "assumptions [(65%)] are tightly coupled to the model of the system under analysis, and when that model changes [..] these assumptions will have to co-evolve" [18]. They also argue that assumptions should clarify the existence of threats and include a rationale. Threat models are also used for risk management [19, 15, 14], as a "threat model provides most of the information necessary for requirements elicitation" [19]. For example, Page et al. derive misuse cases from trust assumptions [15]. They describe trust assumptions with a textual description and related to a use case.

Coding: From this coding phase, we retrieved the connection between assumptions and model elements. In our concept, this is represented with the link to the design decision. We further distinguished between view and view type (as Atkinson et al. [52]). The *view type* is the type of view, e.g., a deployment view on an architectural level. It combines the abstraction level and the information that is viewed at. The *view* is then the instantiated view type. The elements of a view may be influenced by *design decisions*. We did not adapt our concept to threat models as those are certain view types with threats as model elements. However, to describe the impact of the invalidation of the assumption, we considered a risk description as part of an assumption.

Quantification: After surveying 90 papers, Verendel describes assumptions as one of the main artifacts for quantification of security [53]. Moreover, they state: "It is clear that metrics and models for security need validation: using correct assumptions" [53]. They identify four types of assumptions: "that random events in a system occur with *probabilistic independence*", "that [] different agents [] will act rationally", "that quantified system, threat or vulnerability properties are invariant", and "that quantitative information from system components can be composed" [53]. In contrast to them, Chakraborty et al. use hardware assumptions for their quantification over side-channel attacks [16].

Coding: As users and hardware are usually modeled, e.g., as usage or deployment information, assumptions on these are covered by the *design decisions* and related *model elements*. With this in mind, the differentiation between *DesignDecision* and *Decision* becomes redundant.

Others: Besides the requirement elicitation artifacts, Wang et al. identify assumptions as a challenge for the reliability of a machine learning model [30]. They characterize them as hard to

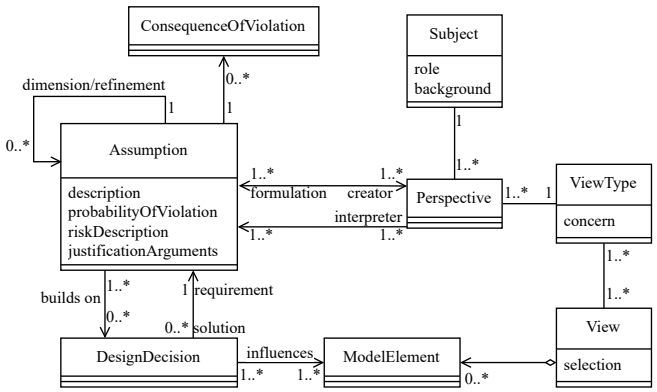

Figure 3. Resulting concept

distinguish from requirements, risks, and constraints of models, context-dependent, and dynamic as they can evolve. Li et al. extract assumptions from mails between software developers using machine learning [54]. For their dataset creation, they use the following criteria: Assumptions are uncertain, uncertainty does not automatically indicate assumptions, assumptions are context-dependent, and there is a difference in the context between assumptions and artifacts, e.g., "the content of a decision focuses on a solution to a problem, while the content on an assumption deals with whether something is correct, suitable, valuable, etc." [54]. In contrast to them, Roeller et al. define: "[design] decisions, as well as the[ir] reasons [..] are implicit, and usually remain undocumented. We call them assumptions" [55]. They recover undocumented design decisions and their rationales using different sources, e.g., interviews, documentation, or source code.

Coding: The works support coding a description for an assumption as mentioned previously in the *context* paragraph. As described by Wang et al. it seems hard to distinguish between design decisions and rationales. This is also confirmed by Roeller et al.. However, we think that the relation between assumption and design decision in our concept covers this, especially, if refined to two relations as explained previously in the *security requirements* paragraph. Even though Li et al. describe the context-dependence of assumptions, they give no overview of relevant context information. Thus, we assume that our concept covers this aspect with its perspective.

2) Requirements Verification: For requirements verification, it is most useful to find conflicts between assumptions and the designed or implemented system. Thereby, the focus is more on the tracing, monitoring, and inconsistency detection between assumptions and the system.

Design: Many works emphasize the linkage of assumptions and architecture and the importance of documenting assumptions for an overview, knowledge management, and traceability [24, 56, 57, 26, 58, 59]. Therefore, assumptions are often described as part, especially as the rationale, of design decisions [56, 59]. However, as assumptions are also used to describe evolution scenarios, variability of product lines, or reusability of architectural parts, they seem to not be part of design decisions but have a high impact on them [7, 21, 60], e.g., in case of Lago and van Vliet (c.f. Section II) [21].

Given the significant impact of context on the system, context assumptions are crucial for its design. Gaaloul et al. synthesize environmental assumptions from components and use them to validate requirements. They understand assumptions as a disjunction of constraints [61]. Similarly, Greenyer et al. formally express behavioral and environmental assumptions to validate requirements. This implies annotations in different kinds of models, e.g., sequence diagrams. There are several other works that use formalized assumptions for the compliance of requirements: assumption-guarantee contracts in temporal logical form [63, 64, 65, 66], assertion-like formalization [23], or impact analyses [67]. Thereby, assumptions are often not distinguished from conditions.

Coding: For the coding, we also considered assumption notions that relate to design (c.f. Section II) [62, 8, 2, 23, 40, 3, 4]. The works support the changes of the previous paragraphs and confirm the connection of assumptions to model elements and their transition to requirements described in the *security requirements* paragraph.

Threat models: In security, the verification of the design can also be coupled with security requirements and threat models. Heyman et al. formalize security requirements as assertions to describe the constraints on the system architecture and identify context assumptions from generated violations [68]. In another work, Heyman et al. propose a framework that supports the reuse of such validated security patterns [69]. Therefore, they distinguish two types of assumptions: "The [trust] assumptions of the refined model document what the pattern requires of its environment to realize its (local) security requirement. [..] The assumptions found by connecting the abstract model to the rest of the system, document what the composition requires to realize its security requirements."

Coding: The perspective-dependent transition between assumption and requirement confirms that those are similar and thereby their relations to design decisions.

Code: Several approaches elicit assumptions from source code [70, 71, 72]. For example, Zschaler and Rashid manually analyze source code to retrieve a classification of aspect-oriented assumptions. They focus on assumptions of aspect developers that have no control over the things outside the aspect. Thus, they retrieve two assumption categories firsthand: "Aspect-Aspect Coordination" assumptions that include "assumptions [that] an aspect A makes on the behavior of other aspects" and "Aspect-Base Coordination" assumptions [71]. The latter cover "assumptions on the relation between aspect and base", e.g., assumptions about synchronization, architectural assumptions, or the internal structure of the base code [71]. If assumptions are found in the code, they can be validated [73, 25, 74]. For example, Goldman and Katz verify the correctness of an aspect [73]. Their "specification includes [formalized] assumptions about properties of the [..] system, and guaranteed properties of any system after the aspect is woven into it".

Coding: The mentioned works demonstrate that a change

of perspective has an influence on the representation of the assumption. Especially, the control aspect of the perspective confirms the relation of the actor and supports the introduction of the view type. Thus, actors may make assumptions about other view types than they have access to. Moreover, Goldman and Katz use the same presentation for assumptions and desired (or required) properties [73]. This again confirms our relations between assumptions and design decisions. Due to term-wise confusion regarding "actor" during our coding, we renamed it to subject, even though its semantic includes systems.

3) General approaches: There are some approaches that address multiple steps in requirements engineering [11, 12, 75, 76, 13, 41, 42, 77, 43, 22, 78]. Haley et al. provide in their works a framework for requirement elicitation and verification based on trust assumptions. Their framework models a recursive project of refinements of goals, requirements, and trust assumptions. To check whether a security requirement is satisfied, they propose *satisfaction arguments* consisting of an outer and an inner argument [76, 42]. To find adequate contexts, they use Jackson's problem frame [9, 76]. In this frame, the outer argument claims a behavior of the system and can be formalized in logical expressions. Thus, it relies on two assumptions: a correct context, and no additional conflicts in the implementation. The inner argument, inspired by Toulmin [51], supports the outer argument and is usually informal. With Toulmin's technique, underlying assumptions can be exposed and tested on veracity [42]. Thus, assumptions can be invalid if they do not hold themselves or if they depend on deeper unstated assumptions that are incorrect [42].

Tirumala [22] traces assumptions across the software development life-cycle. They annotate assumptions and guarantees to components and describe their composition, including vertical AM. However, in contrast to the works of Haley et al., they offer no verification at the architectural level.

Coding: In this coding we also considered Viega et al. (c.f. Section II) [10]. The approaches state that assumptions can be layered and connected to other assumptions. Moreover, they define their perspectives according to the used view type. In conclusion, the works confirmed our concept and indicated no needed changes.

C. Resulting Concept and Definition

Figure 3 shows the conceptual model that results from the CGT-based approach. Even though we collected new insights during the literature review, these were always confirmed by multiple works: The *Assumption* was extended by a *description*, *justificationArguments*, and a *riskDescription* that describes the impact of the invalidation of the assumption. Thus, the *Criticality* was resolved. To keep the different types of consequences, we introduced a *ConsequenceOfViolation*. The human *Actor* of the *Perspective* was replaced by *Subject* to extend the term to systems. The *levelofAbstraction* was replaced by the *ViewType* as part of the *Perspective*. As *ViewType* covers *modelLevel* and *informationLayer*, these concepts were removed. The *View* now encapsulates different *ModelElements* that can be influenced by *DesignDecisions*. *DesignDecisions* have two types of relations: A requirement-solution relation for *DesignDecisions* that solve assumptions, and a *based on* relation for *DesignDecisions* that build on assumptions. In the first case, assumptions are not different from requirements. In the second case, assumptions can be similar to a rationale.

Based on that, we formulate the definition to answer RQ1:

Definition 1. A security-related assumption is a statement that:

1) is taken for true,
2) can be violated,
3) is formulated from a certain perspective [1]

Security-related assumptions have further properties:

4) they can be interpreted from different perspectives
5) they can be refined horizontally (on the same abstraction level), as well as vertically (across diff. abstraction levels),
6) they have a probability to be violated, they have a risk for the security of the system, as well as consequences of their violation,
7) they occur when a design decision is made, and
8) they transform into a requirement if they are supposed to be realized by a design decision.

V. Evaluation

For our evaluation, we conducted two questionnaires: One questionnaire about security-related assumptions and another about general assumptions that occur during the software development process. Except for the renaming of "security-related assumptions" into "assumptions", the structure and questions were similar in both cases. We conducted the studies on LimeSurvey [79] and used Prolific [80] for participant acquisition. The questionnaire, its resulting data, and graphics can be found in our replication package [37]. Our process and description adheres to the ACM Empirical Standards [81]. Thus, we mention references, e.g., (5a), that relate to specific questions of our questionnaire and their results. We report results for both questionnaires with a notion that uses pairs that shows the results for security-related assumptions first. For example, "10-5%" means the answer was selected by 10% of the participants of the security-related assumptions questionnaire and by 5% of the questionnaire about general assumptions in software development.

A. Questionnaire Design

The questionnaire was designed using several guidelines [82, 83] and is reported regarding the ACM empirical standards [81]. The landing page mentions the goal, a completion time of about 5 min as well as the conductors of the survey. The next page is a short introduction that introduces and explains the terms *requirements*, *decisions*, and *perspective*. Further, the definition of a (*security-related*) *assumption* is introduced. The next survey section records the participant population by profession, confidence regarding relevant software engineering domains as well as the frequency of different activities with requirements, design decisions, and security features. To answer

[1] A perspective characterizes a view that a certain subject has based on a certain view type, e.g., a software architecture or uml class model

RQ2, the participants are further asked to rate twelve statements about properties of (security-related) assumptions using a five-point Likert scale. This part also includes one statement as an attention check that clearly contradicts the provided definition. In the same style, we included questions to answer RQ3. Thus, the participants are asked to rate nine statements about the documentation of (security-related) assumptions. Finally, they can leave some feedback, provide their email address to receive their study results, and be thanked for their participation.

B. Survey Execution

For the execution, we requested participants with knowledge of software development techniques from IT/ Information Networking/ Information Security, Engineering (e.g. software) jobs, and companies with at least 250 employees. We further restricted the residence of potential participants to lie in one of the 19 sovereign countries comprised by G20. For better global distribution, we started 50% of each survey 12 hours apart (at 4 am resp. 4 pm UTC) on working days. As a reward, we payed minimum 9 £ per hour.

After the execution of the surveys, we reviewed the results. The mean time of all participants was between 5-6 min. We rejected all surveys that missed the attention check or took less than 1:30 min. Due to a misconception, we have participants who took part multiple times in a survey. Similar to participants who answered both surveys, we keep the results of the first survey. We also remove surveys that are answered in less than three minutes as the survey cannot be answered meaningfully in such a short time frame.

We obtained 81 valid answered surveys for security-related assumptions and 67 for general assumptions in software development. Overall, the participants come from at least ten countries and 16 nationalities, mainly UK (23%), South Africa (21%), the US (13%), and Germany (10%). Most identify as male (68%, 26% women) and are around 30 years old. 61% of the participants work in companies that are larger than 1000+ employees, whereas 32% of participants work in companies with 250 to 999 employees. In both studies, software engineers form the biggest group (35-40%), software developers second biggest (26-22%), and others forming the third biggest group (17-18%). Regarding the expertise, the participants of both groups answer to be highly confident in all categories. Participants are most confident with the software architecture domain with more than 70% participants stating they are at least confident (74-79%, neutral: 20-13%). The participants rate their expertise in software architecture documentation similarly (overall agreement: 72-75%, neutral: 20-21%). In requirements engineering, more than 60% of all participants are at least confident with their abilities (neutral: 27-22%). In software security, confidence dropped to roughly 50% with a third (31-37%) having a neutral mindset.

C. Results

Regarding security-related assumptions, we observe a strong agreement on most properties.

The scope for interpretation (4) is overall agreed (84-81%).

The refinement property (5) is split up into four statements regarding vertical (5a, 5b) and horizontal refinement (5c, 5d). Most participants in both groups agree with possible vertical refinement (88-82%, neutral: 10-16%). The statement that vertically refined assumptions are valid if and only if the fine-grain assumptions are valid (5b) is differently distributed in both groups. For security-related assumptions, 37% vote neutral, 40% agree and 16% strongly agree. For general assumptions in software development, the participants have a shift towards higher agreement (21% neutral, 45% agree, 25% strongly agree). In contrast, only 56% agree on horizontal refinement (5c) for security-related assumptions and 66% for general assumptions. However, a much bigger share (10-12%) reject (5c) compared to (5a) (1-2%). Regarding the validity of vertically refined assumptions (5d), both kinds have similar distributions: 70% at least agree with the statement whereas few (6-7%) disagree.

The sixth property is divided into three sub-statements (6a, 6b, 6c). The participants agree much more with the quantifiability (6a) in the case of security-related assumptions (overall 64-43%) and disagree much less with the property (overall 7-24%). For both kinds of assumptions, the participants majorly agree (75-70%) with the risk to the (security of a) system (6b) and few (7-9%) disagree. Similarly for (6c), there is an overall high agreement (84-87%) that assumptions can cause consequences (disagreement: 4-3%).

Property (7), the design decisions as an origin of assumptions, is represented by statement (7a). Many (62-66%) overall agree with this statement (disagreement: 9-15%). Participants are more convinced with statement (7b) which suggests the dependency of design decisions on assumptions (agreement 81-76%; disagreement: 5-1%).

The relation of requirements and assumptions (8) is reflected in four statements. The dependency of a requirement on an assumption (8a) is mainly accepted in both cases (overall 80-84%) while few (6-1%) disagree. Similarly, the participants also overall agree (85-82%) on the other way round (8b; disagreement: 1-3%). Whether design decisions can be made to ensure assumptions (8c) is rated slightly differently in both groups. For security-related assumptions, 60% agree, 23% strongly agree, whereas no participant disagree. Regarding general assumptions in software development, only 46% agree, 15% strongly agree, and 6% disagree. In contrast, the transformation of assumptions into requirements (8d) is mostly supported in both groups (overall 73-70%), with 25% in each group being indifferent.

Regarding the statements on the documentation of assumptions, the participants only slightly disagreed (overall disagreement $<5\%$) with one of the statements (D5). All statements regarding the documentation had a higher amplitude to strong agreements for security-related assumptions, i.e., both ratios often differ by more than 10 percentage points. Almost all (98-94%) agree that assumptions should be documented (D1). For security-related assumptions, 58% even strongly agree. Both groups also agree on documenting the impact of an assumption (D2; overall 93-87%). The participants also agree

that the role (D3) and the view type (D4) of the formulator should be included in the documentation (overall D3: 77-73%, D4: 77-79%). The documentation of the probability of a violation (D5) is less agreed upon compared to the other properties (overall agreement: 77-63%, disagreement: 7-12%). In contrast, the participants overall agree with the documentation of consequences (D6; 88-79%), impact (D7; 91-85%), the reasoning for the validity (D8; 81-84%), and trace links to the system (D9; 81-81%).

D. Discussion

The results of our questionnaire support our motivation that assumptions are important for reliable software development and should be documented.

The survey confirmed most properties of (security-related) assumptions. Yet, the statement about the occurrence of assumptions when design decisions are made has comparatively many disagreements (15% in the case of general assumptions). Maybe, some examples would have helped with understanding the statements, e.g. on horizontal refinement.

The participants clearly agree upon the transformation of assumptions into requirements. Further, trace links to other artifacts should be documented like in requirements engineering. This similarity between requirements and assumptions is a remarkable insight and indicates that existing approaches from requirements engineering might be reusable for AM.

Regarding RQ2, neither our theory building nor our validation could identify a concise difference between security-related and general assumptions in software development. The only difference seems to be the probability of violation. Nevertheless, both groups agree that consequences, impact, and reasoning of assumptions should be documented.

Our survey provides a minimal subset for the documentation of assumptions and, thereby, answers RQ3. Except for the probability of violation for general assumptions in software development the documentation of (security-related) assumptions and their attributes was recommended in the survey.

VI. Threats to Validity

In this section, we discuss qualitative criteria from the ACM empirical standards [81] and threats to validity based on guidelines from Runeson and Höst [84].

a) Credibility: Our theory could contain a bias due to its origin from a small number of interviews from a specific domain and location. To compensate for this, we used publications on AM to further strengthen the theory. We finally validated the theory with 148 globally distributed participants to have multivocal evidence for our theory [81].

b) Usefulness: This paper provides a definition for security-related assumptions, extends it to general assumptions in software development, and validates it. Despite the use of literature during theory-building, there is the risk that the definition is not useful for assumption management or requirements engineering. However, our survey results indicate the applicability of existing approaches from requirements engineering to AM, demonstrating a certain usefulness.

c) Resonance: Our results from the questionnaire provide a positive resonance to the given theory. Nevertheless, the resonance might be affected by a conformity bias as most survey participants agreed with most statements.

d) Construct & Internal Validity: There is the risk of subject-expectancy effects in our study. To mitigate this, we used a fixed-question template and predefined time intervals during the interviews. Additionally, only one interviewer was assigned to minimize confirmation bias. Moreover, two fixed researchers conducted the memoing and coding process for the interviews, further minimizing negative effects.

The theory-building process could also be affected by missed literature. However, as the theory did not change at the end of the literature review, the effect is assumed small and we further reduced it with our theory validation, i.e., our survey.

The payment, time, order of questions, as well as description of the introduction of our questionnaire, could have affected the internal validity of the study. To diminish the effects, we paid the recommended money per hour, kept the questionnaire short, sorted questions by properties, and tried to make the description as clear as possible, including examples. Despite that, the formulation of statements is a threat to validity.

e) External Validity: Even though the provided theory was mainly formulated for security-related assumptions, it seems mostly applicable to general assumptions in software development: The general literature on AM used in theory-building as well as the conducted survey indicate the generalizability of large parts of the definition. However, we and our participants could still have missed some restrictions.

f) Reproducibility & Reliability: For reproducibility, we asked the interviewees for their agreement to share their manuscripts. All but one agreed and all these manuscripts can be found in our replication package [37]. Yet, the theory-building process is a creative process and difficult to replicate exactly. The package also includes all survey results, a script for evaluation, the analyzed data and bar charts for every question, and a script to generate them. The reliability could also be affected by the small number of interviews. We lowered the effect with the extensive literature review and survey.

VII. Conclusion & Future Work

There is substantial impact of security-related assumptions on system design, implementation, and security. Explicit assumption management facilitates identifying, describing, evaluating, maintaining, tracing, monitoring, reusing, and organizing a system's assumptions.

This paper defines security-related assumptions in software development. The definition is derived using a Constructivist Grounded Theory-based approach, which involves literature analysis and structured interviews in academia. We further validated the definition using two surveys. Our results indicate that the definition is mostly applicable to general assumptions in software development. The results further confirmed that the documentation of assumptions is important and should include several features. Finally, we could show that assumptions

can be transformed into requirements and, thus, have a tight relationship and similarities.

As a result, this work lays a foundation for reusing existing approaches from requirements engineering in assumption management during software development. Moreover, its definition supports future endeavors in automating assumption management, estimating system security, and quantifying security measures.

REFERENCES

[1] P. Zave and M. Jackson, "Four dark corners of requirements engineering," *ACM Trans. Softw. Eng. Methodol.*, 1997.

[2] M. A. A. Mamun and J. Hansson, "Review and challenges of assumptions in software development," in *Second Analytic Virtual Integration of Cyber-Physical Systems Workshop (AVICPS)*, 2011.

[3] C. Yang *et al.*, "Assumptions and their management in software development: A systematic mapping study," *Information and Software Technology*, 2018.

[4] C. Yang *et al.*, "Architectural assumptions and their management in industry – an exploratory study," in *Software Architecture*, 2017.

[5] I. Kabanov and S. Madnick, "A systematic study of the control failures in the equifax cybersecurity incident," 2020.

[6] D. Parnas, "Information distribution aspects of design methodology," 1971.

[7] D. Garlan *et al.*, "Architectural mismatch: why reuse is so hard," *IEEE Software*, 1995.

[8] M. Lehman and J. Fernandez-Ramil, "Software evolution in the age of component-based software engineering," *Software, IEE Proceedings -*, 2000.

[9] M. Jackson, *Problem frames: analysing and structuring software development problems*. Addison-Wesley, 2001.

[10] J. Viega *et al.*, "Trust (and mistrust) in secure applications," *Commun. ACM*, 2001.

[11] C. B. Haley *et al.*, "Using trust assumptions in security requirements engineering," in *Second Internal iTrust Workshop On Trust Management In Dynamic Open Systems*, 2003.

[12] C. Haley *et al.*, "The effect of trust assumptions on the elaboration of security requirements," in *Proceedings. 12th IEEE International Requirements Engineering Conference, 2004.*, 2004.

[13] C. B. Haley *et al.*, "Using trust assumptions with security requirements," *Requirements Engineering*, 2006.

[14] ISO, "Iso/iec 15408-1: 2009 information technology—security techniques—evaluation criteria for it security—part 1: Introduction and general model," *Int. Organ. Stand*, 2009.

[15] V. Page *et al.*, "Security risk mitigation for information systems," *BT Technology Journal*, 2007.

[16] P. Chakraborty *et al.*, "Haste: Software security analysis for timing attacks on clear hardware assumption," *IEEE Embedded Systems Letters*, 2022.

[17] D. Van Landuyt and W. Joosen, "A descriptive study of assumptions made in linddun privacy threat elicitation," in *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, 2020.

[18] D. Van Landuyt and W. Joosen, "A descriptive study of assumptions in stride security threat modeling," *Software and Systems Modeling*, 2021.

[19] S. Myagmar *et al.*, "Threat modeling as a basis for security requirements," in *Symposium on Requirements Engineering for Information Security (SREIS)*, 2005.

[20] H. El-Hadary and S. El-Kassas, "Capturing security requirements for software systems," *Journal of Advanced Research*, 2014.

[21] P. Lago and H. van Vliet, "Explicit assumptions enrich architectural models," in *Proceedings. 27th International Conference on Software Engineering, 2005. ICSE 2005.*, 2005.

[22] A. S. Tirumala, "An assumptions management framework for systems software," Ph.D. dissertation, University of Illinois at Urbana-Champaign, 2006.

[23] M. A. Mamun *et al.*, "Towards formalizing assumptions on architectural level: A proof-of-concept," Chalmers University of Gothenburg, Tech. Rep., 2012.

[24] C. Yang *et al.*, "An industrial case study on an architectural assumption documentation framework," *Journal of Systems and Software*, 2017.

[25] G. Lewis *et al.*, "Assumptions management in software development," Carnegie Mellon University, Tech. Rep., 2004.

[26] H. Ordibehesht, "Explicating critical assumptions in software architectures using aadl," Gothenburg, Sweden, 2010.

[27] B. GLASER, "The discovery of grounded theory," *Strategies for Qualitiative Research*, 1967.

[28] K.-J. Stol *et al.*, "Grounded theory in software engineering research: a critical review and guidelines," in *Proceedings of the 38th International conference on software engineering*, 2016, pp. 120–131.

[29] K. Charmaz, *Constructing grounded theory*, ser. Introducing Qualitative Methods Series. SAGE Publications, 2014.

[30] S. Wang *et al.*, "Synergy between machine/deep learning and software engineering: How far are we?" *CoRR*, 2020.

[31] J. A. Dewar *et al.*, *Assumption Based Planning: A Planning Tool for Every Uncertain Times*. Rand, 1993.

[32] H. Enquist and N. Makrygiannis, "Understanding misunderstandings [in complex information systems development]," in *Proceedings of the Thirty-First Hawaii International Conference on System Sciences*, 1998.

[33] K. Katkalov *et al.*, "Model-driven development of information flow-secure systems with iflow," in *2013 International Conference on Social Computing*, 2013.

[34] S. Dolnicar, "Asking good survey questions," *Journal of Travel Research*, vol. 52, pp. 551–574, 2013.

[35] F. Shull *et al.*, *Guide to Advanced Empirical Software Engineering*. Springer-Verlag, 2007.

[36] E. Knott *et al.*, "Interviews in the social sciences," *Nature Reviews Methods Primers*, vol. 2, p. 73, 2022.

[37] Anonymous, 2024. [Online]. Available: https://figshare.com/account/articles/25061507

[38] P. Van Aubel and E. Poll, "Smart metering in the netherlands: What, how, and why," *International Journal of Electrical Power & Energy Systems*, 2019.

[39] F. J. Fowler, *Improving survey questions: Design and evaluation*. Sage, 1995.

[40] C. Yang *et al.*, "A survey on software architectural assumptions," *Journal of Systems and Software*, 2016.

[41] C. B. Haley *et al.*, "A framework for security requirements engineering," in *Proceedings of the 2006 international workshop on Software engineering for secure systems*, 2006.

[42] C. Haley *et al.*, "Security requirements engineering: A framework for representation and analysis," *IEEE Transactions on Software Engineering*, 2008.

[43] B. Nuseibeh *et al.*, "Securing the skies: In requirements we trust," *Computer*, 2009.

[44] C. Curtain. (2023, Jan.) QualCoder. [Online]. Available: https://github.com/ccbogel/QualCoder/releases/tag/3.2

[45] M. Jackson, "Problem frames and software engineering," *Information and Software Technology*, 2005.

[46] B. Broadnax *et al.*, "Eliciting and refining requirements for comprehensible security," in *Security Research Conference : 11th Future Security, Berlin, September 13-14, 2016. Ed.: O. Ambacher*, 2016.

[47] X. Wang *et al.*, "How software changes the world: The role of assumptions," in *2016 IEEE Tenth International Conference on Research Challenges in Information Science (RCIS)*, 2016.

[48] A. Bazarhanova and K. Smolander, "The review of non-technical assumptions in digital identity architectures," in *Proceedings of the 53rd Hawaii International Conference on System Sciences*, 2020.

[49] A. Bazaz *et al.*, "Modeling security vulnerabilities: A constraints and assumptions perspective," in *2006 2nd IEEE International Symposium on Dependable, Autonomic and Secure Computing*, 2006.

[50] S. Faily and I. Fléchais, "The secret lives of assumptions:

Developing and refining assumption personas for secure system design," in *Proceedings of the Third International Conference on Human-Centred Software Engineering*, 2010.
[51] S. E. Toulmin, *The Uses of Argument*. Cambridge University Press, 1958.
[52] C. Atkinson *et al.*, "Orthographic software modeling: a practical approach to view-based development," in *International Conference on Evaluation of Novel Approaches to Software Engineering*, 2008.
[53] V. Verendel, "Quantified security is a weak hypothesis: A critical survey of results and assumptions," in *Proceedings of the 2009 Workshop on New Security Paradigms Workshop*, 2009.
[54] R. Li *et al.*, "Automatic identification of assumptions from the hibernate developer mailing list," in *2019 26th Asia-Pacific Software Engineering Conference (APSEC)*, 2019.
[55] R. Roeller *et al.*, "Recovering architectural assumptions," *Journal of Systems and Software*, 2006.
[56] P. Kruchten *et al.*, "Building up and reasoning about architectural knowledge," in *Quality of Software Architectures*, 2006.
[57] J. Tyree and A. Akerman, "Architecture decisions: demystifying architecture," *IEEE Software*, 2005.
[58] D. Van Landuyt and W. Joosen, "Modularizing early architectural assumptions in scenario-based requirements," in *Fundamental Approaches to Software Engineering*, 2014.
[59] T. Dingsøyr and H. van Vliet, *Introduction to Software Architecture and Knowledge Management*. Springer Berlin Heidelberg, 2009.
[60] A. Miranskyy *et al.*, "Modelling assumptions and requirements in the context of project risk," in *13th IEEE International Conference on Requirements Engineering (RE'05)*, 2005.
[61] K. Gaaloul *et al.*, "Mining assumptions for software components using machine learning," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020.
[62] J. Greenyer *et al.*, "The scenariotools play-out of modal sequence diagram specifications with environment assumptions," *Electron. Commun. Eur. Assoc. Softw. Sci. Technol.*, 2013.
[63] T. Arts *et al.*, "Making implicit safety requirements explicit," in *Computer Safety, Reliability, and Security*, 2014.
[64] A. Cimatti *et al.*, "Ocra: A tool for checking the refinement of temporal contracts," in *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2013.
[65] R. Seater *et al.*, "Requirement progression in problem frames: deriving specifications from requirements," *Requirements Engineering*, 2007.
[66] J. Sun *et al.*, "Specifying and verifying event-based fairness enhanced systems," in *Formal Methods and Software Engineering*, 2008.
[67] A. Tang *et al.*, "A rationale-based architecture model for design traceability and reasoning," *J. Syst. Softw.*, 2007.
[68] T. Heyman *et al.*, "Security in context: Analysis and refinement of software architectures," in *2010 IEEE 34th Annual Computer Software and Applications Conference*, 2010.
[69] T. Heyman *et al.*, "Reusable formal models for secure software architectures," in *2012 Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture*, 2012.
[70] J. P. Near *et al.*, "A lightweight code analysis and its role in evaluation of a dependability case," in *2011 33rd International Conference on Software Engineering (ICSE)*, 2011.
[71] S. Zschaler and A. Rashid, "Aspect assumptions: A retrospective study of aspectj developers' assumptions about aspect usage," in *Proceedings of the Tenth International Conference on Aspect-Oriented Software Development*, 2011.
[72] J. Fabry *et al.*, "Aspectual source code analysis with gasr," in *Proceedings of the 13th International Working Conference on Source Code Analysis and Manipulation (SCAM 2013)*, 2013.
[73] M. Goldman and S. Katz, "Maven: Modular aspect verification," in *Tools and Algorithms for the Construction and Analysis of Systems*, 2007.
[74] S. Fickas and M. Feather, "Requirements monitoring in dynamic environments," in *Proceedings of 1995 IEEE International Symposium on Requirements Engineering (RE'95)*, 1995.
[75] C. B. Haley *et al.*, "Picking battles: The impact of trust assumptions on the elaboration of security requirements," in *Trust Management: Second International Conference, iTrust 2004, Oxford, UK, March 29-April 1, 2004. Proceedings 2*, 2004.
[76] C. B. Haley *et al.*, "Arguing security: validating security requirements using structured argumentation," in *Third Symposium on Requirements Engineering for Information Security (SREIS'05) held in conjunction with the 13th International Requirements Engineering Conference (RE'05)*, 2005.
[77] C. B. Haley and B. Nuseibeh, "Bridging requirements and architecture for systems of systems," in *2008 International Symposium on Information Technology*, 2008.
[78] T. T. Tun *et al.*, "Model-based argument analysis for evolving security requirements," in *2010 Fourth International Conference on Secure Software Integration and Reliability Improvement*, 2010.
[79] LimeSurvey Project Team / Carsten Schmitz, *LimeSurvey: An Open Source survey tool*, LimeSurvey Project, Hamburg, Germany, 2012. [Online]. Available: http://www.limesurvey.org
[80] Prolific, *Prolific*, Prolific, London, UK, 2014. [Online]. Available: https://www.prolific.com
[81] P. Ralph *et al.*, "ACM SIGSOFT empirical standards," *CoRR*, 2020.
[82] H. Taherdoost, "Designing a questionnaire for a research paper: A comprehensive guide to design and develop an effective questionnaire," *Asian Journal of Managerial Science*, 2022.
[83] G. Marshall, "The purpose, design and administration of a questionnaire for data collection," *Radiography*, 2005.
[84] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Software Engineering*, 2009.