Software Security Specifications and Design

How Software Engineers and Practitioners Are Mixing Things up

Mohammad Zarour
Software Engineering Department
Prince Sultan University, Riyadh,
Saudi Arabia
mzarour@psu.edu.sa.

Mamdouh Alenezi
Software Engineering Department
Prince Sultan University, Riyadh,
Saudi Arabia
malenezi@psu.edu.sa

Khalid Alsarayrah
Software Engineering Department
Hashemite University, Zarka,
Jordan
khalidt@hu.edu.jo

ABSTRACT

Huge numbers of worldwide-deployed software suffer from poor quality and possess vulnerabilities with serious impact. Meanwhile, people are using such software to save and manage their valuable information including their monetary data. This has increased the hackers' appetite to attack software. Henceforth, researchers and practitioners are convinced that software security is not an added value or a gold-plating need. Consequently, security requirements specification and implementation become vital during the software development process. Unfortunately, researchers and practitioners are doing so in a rush. This has made them mix concepts and practices up in a way that can terribly make the problem of delivering software overdue more chronic which will result in a security and technical debt. This research represents a corrective study that sheds light on what has been achieved in analyzing and designing secure software and what are the problems committed and how to handle them.

CCS CONCEPTS

• Software and its engineering • Software creation and management

KEYWORDS

Software Architecture, Software Design, Software Development Management, Software Engineering, Security

ACM Reference format:

Mohammad Zarour, Mamdouh Alenezi and Khalid Alsarayrah. 2020. Software Security Specifications and Design: How Software Engineers and Practitioners Are Mixing Things up. In Proceedings of ACM International Conference on Evaluation and Assessment in Software Engineering (EASE'20). ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3383219.3383284

1 Introduction

Since the late sixties of the previous century, where the term software engineering is coined, software engineers are striving to develop software that meets customer needs, represented as functional and non-functional requirements, within time and budget constraints. Many software processes models have been developed as our understanding of the software nature and its needs are developed. The focus was, and still be, on delivering what the customers expect from the software to run their businesses and achieve their functionalities within the traditional quality needs related to efficiency, reliability, and 'naïve' security.

Nowadays, 'naïve' software security that focuses on securing software users by password-protected accounts with certain privileges is not enough. We continuously discover that a big number of deployed software worldwide is of poor quality and suffering from vulnerabilities with serious impact [1]–[4]. The wide-spread usage of the internet, huge web-based and mobile apps development along with the unprecedented amount of data generated on daily bases that comprise all our life aspects, have increased the hackers' appetite to attack software. Hackers have been increasingly exposing and exploiting vulnerabilities for a long time, and their success is very probable nowadays due to the software poor quality. Unfortunately, traditional network perimeter defenses that include firewalls, antiviruses, and intrusion detection and prevention systems have, to a large extent, failed to stop software attacks. This is because hackers are focusing more on attacking software leveraging from undisclosed vulnerabilities. Hence, the ball has been thrown back to the software development community to produce more secure applications using secure development process models.

Consequently, several initiatives have been made to address security in the software development lifecycle (SDLC). This includes models from industry, such as Microsoft Security Development Lifecycle (SDL) [5], descriptive activity surveys such as Building Security in Maturity Model (BSIMM) [6], and standards, such as the ISO/IEC 27034 [7]. Moreover, security initiatives and tools that support the integration of security in the development lifecycle have been proposed by different researchers, see for example [8]–[11]. Unfortunately, despite of all these efforts to tackle the security requirements during the development process, vulnerable software products are still produced and successful attacks never stop. This means that the way the security requirements are captured during the requirement and design phases may not be suitable and need further investigation. This is what we try to examine on this research work. Software security is the idea of engineering software so that it continues to function correctly under malicious attack [12]. This means to adopt software engineering practices in the software process that tackle security issues early. The rest of the paper is organized as follows. Section 2 presents a literature review about software security and the related work. Section 3 discusses how researchers and practitioners are mixing things up. Section 4 discusses the security in the requirements engineering phase while section 5 discusses the security in the design phase. Section 6 discusses possible limitations. Section 7 concludes the paper and presents future work.

2 Literature Review

Software is being developed for decades until now. Software engineers and IT professionals were able to develop software that meets customers' requirements. Security was not an issue in the past as most of the developed applications were standalone applications with very restricted connectivity. Hence, the software development process focused on developing software that is functional with high performance but not necessarily secure. Nowadays, with the advent of the Internet of Things (IoT) and ubiquitous computing, software engineers are developing software that is more complex with higher extensibility and connectivity features using the traditional development lifecycles that ignore the software security as a demanding and desirable requirement. Software extensibility and connectivity effects on security has been discussed in [13]. The new technologies help to evolve better software, but hackers and hacking strategies are evolving much faster as well. Henceforth, security is not considered as an added value anymore. Accordingly, software processes have been amended to integrate secure software development practices in what is known as secure software development. Developing security conscious software to face the increasing amount and quality of cyber-attacks is still lacking maturity and the research in this domain is still in its infancy phase, hence we are conducting an ordinary literature review to cover what has been documented in the literature. Although developers are not necessarily security experts, they are expected to develop secure software [14] and are held responsible for discovered security vulnerabilities [15]. Unfortunately, the research work that discusses the human effects on software security is lacking and the weakest link in such software development are the developers! [16][17].

But developers lack necessary security knowledge which resulted in loose security practices [17]. Moreover, security is not considered in the design stage, security is not a priority during implementation, developers do not test for security, and security is not considered during code review [17]. Hence, it is unrealistic to rely on developers only to perform security tasks while lacking the expertise [17]. Developers are usually focused on achieving the specified functionalities and performance requirements and security is not their main concern [16]. Some researchers proposed a solution to this problem by taking the secure code development from developers and assign it to analysis tools and specialized easy to use APIs, or at least use these analysis tools and APIs to provide help and guidance to developers while developing secure software [18]–[20]. Blaming developers for vulnerabilities does not solve the problem which extends up in the organizational hierarchy that comprises various issues influencing software security [15]. The focus of this research does not extend to the organizational, cultural and awareness issues, but rather focuses on security issues within the requirements and design phases.

2.1 Security in the Requirements Engineering Phase

Requirements engineering is concerned with discovering, developing, tracing, analyzing, qualifying, communicating and managing requirements that define the system at successive levels of abstraction [21]. Requirements at this phase should be specified independently of any technology/platform. This technology independence is vital in order not to limit the solution space from which the designer designs the software blueprint.

The software is successful when it maintains all the functionalities requested by the users and are expressed as functional user requirements (FUR) and within certain constraints specified as non-functional requirements (NFR). For example, "The registration system shall allow students to register a course" is a FUR, while "The registration system shall encrypt student's credentials when authenticating him/her" is an NFR. The FUR and NFR are specified at the system level as high-level requirements. However, there is yet no consensus on how to describe, specify, measure and evaluate NFR during the early phases, leading to various difficulties and ambiguities [22], this makes specifying security requirements more difficult as well.

Security requirements at the system level are defined as the confidentiality, integrity, availability, and authenticity of the systems [23]. Researchers have documented different approaches to identify, in more details, how to capture such requirements. For instance, McDermott and Fox [24] introduced the concept of Abuse cases to model harmful activities between the system and malicious actors. Abuse case models are claimed to increase both user and customer understanding of the security features of a proposed product. Other researchers followed the same abuse model to analyze security requirements in the requirement phase, see for example, [25], [26]. The concept of misuse case model is introduced by Alexander [27] where the misuse case is defined as a use case from the point of view of an actor hostile to the system. Alexander stated that the interplay of use and misuse cases during analysis could help engineers elicit and organize requirements more effectively. Other researchers have used and extended the concept of misuse case model. For instance, Sindre et. al. [28] have represented both use cases and misuse cases in a single diagram. In addition, they described a detailed template to specify misuse cases. Similarly, Mai et. al. [29] used misuse cases to model security and privacy requirements. Sindre also developed mal-activity swim-lane diagrams [30] as a technique for capturing attacks that could complement misuse cases for early elicitation of security requirements. His technique allowed the inclusion of hostile activities together with legitimate activities in one diagram.

Schmitt and Liggesmeyer [31] proposed a model for Structuring and Reusing Security Requirements during the requirements engineering phase that requires exploring the possible threats, weaknesses, and vulnerabilities as sources that necessitate the security requirements specification during the requirements engineering phase.

Fletcher and Liu [32] recognized that security requirements analysis during the requirements phase is not enough and they presented more security requirements analysis but in the cyber-physical system. Their approach analyzes security requirements by extending activity swim-lane diagrams to include mal-activities and prevention or mitigation options in the same diagram to identify threats posed by both internal and external misusers.

2.2 Security in the Design Phase

The software design phase is where the design decisions are made to build the solution blueprint that fulfills the specified requirements resulted from the requirements engineering phase. The design phase is technology dependent; hence, the designers need to make decisions regarding which technology/platform to use for the system on hand. Designers need to analyze the possible threats that may attack the system based on the chosen technology/platform. Unfortunately, most of the IT practitioners indicated that their development teams did not view security as part of the design phase, or at best security is managed in an ad-hoc fashion during this phase [17]. Assal and Chiasson empirical study [17] concluded that many practitioners do not follow simple design principles and are intentionally introduce complexity to avoid rewriting existing code. Not adopting design principles will generate a complex system design full of architectural flaws that will lead to security problems [33]. Moreover, efforts towards evaluating security may be hindered by poor readability and complex design choices [17].

Nowadays, different levels of security are considered crucial in almost all system. Researchers and practitioners are studying and implementing security at different phases including the design phase, see for example [17], [19], [37]–[40], [22], [25], [27], [29], [32], [34]–[36]. Other researchers have worked on extending the UML notation to represent security concerns, see for example [40]–[44]. But what about systems that have been developed long time ago where security requirements were not treated as a core characteristic, this issue has been raised by Shin and Gomaa [45] where they proposed a solution in which the security requirements are captured, as a separate service, independently from application requirements. Shin and Gomaa have adopted the concept of separation of concern to separate the application concerns from security concerns, then, evolve from a non-secure application to a secure application is achieved.

Security has also been studied at the detailed design level, where various security design patterns have been proposed based on their traditional counter-patterns, see for example [34], [36], [46], [47].

3 How Researchers and Practitioners Are Mixing Things Up?

As discussed in the previous section, the human effect on secure software development is unneglectable. Developers are usually held responsible for vulnerabilities, at least by the end-users, and are expected to develop secure software. We believe that it is not the developers who are the weakest link in the development process, requirement engineers and designers should be equally held responsible as well for any security flaw that leads to vulnerabilities.

Unfortunately, the rush toward developing secure software resulted in mixing tasks and practices to be done by the requirement engineer, designer, and developers. Furthermore, real-life security practices are deviating from best practices identified in the literature [17]. Best practices are often ignored, simply since compliance would increase the burden on the development team [17]. Regrettably, ignoring compliance with standards will not eliminate the burden on the development team, but it will distribute it over the different phase which will result in having delayed software projects be more delayed and will increase the technical debt as well [48]. Moreover, some new terminologies that are extracted from well-known ones have been improperly crammed into the requirement or design context. For instance,

1. The users and stakeholders are involved in developing the proposed abuse/misuse case model and mal-activity swim-lane diagrams. However, how can the normal user help in drawing abuse cases while they are neither experts in security technicalities nor the threats that may affect their proposed system. Eliciting security requirements from stakeholders is hard and can yield partial and skewed results [49]. Hence, the research documented in the literature that elicit detailed-level security requirements from users and stakeholders perspectives, as in [28], [30], [31], [35], [37], [50], are expected to achieve partial success in this regard. Users and stakeholders can express their high-level security needs as functional and nonfunctional requirements, but they are unaware of the details of how their system can be misused or threatened once it is put in use. Hence, cramping abuse/misuse cases at the requirement phase is not effective and can confuse the stakeholders and end up with more vague requirements that need rework in the design phase. Again, this would make the problem of project delivery overdue more chronic which will result in a security and technical debt [48]. It seems that researchers working on security requirements specification are snubbing this issue.
2. Moreover, not all known threats are applicable to all systems. The threat that may attack a certain system depends on the proposed solution, which will be developed during the design phase, not the requirement phase. some researchers suggested engaging the stakeholders in the security design as well [38]. Again, normal stakeholders are naïve when it comes to discussing detailed security threats and those who commit the attacks, i.e. adversaries/hackers, are absent from the whole scene of software development.
3. The use of the term ‘abuse/misuse case’ is misleading by itself. It gives the indication that vulnerabilities will be abused/misused by users, remember that the concept of use

cases is traditionally linked to the possible uses of the system by users. Therefore, when drawing the abuse/misuse cases, the most important actor, i.e. the adversary/hacker, remains unavailable. This means that the identified threats via abuse cases can be explored only indirectly and, at best, specified partially [49].

4. Thereafter, the term threat modeling and its corresponding practices seem to be more suitable to be used rather than the term abuse/misuse cases. Note that threat models are useful for designers more than requirement engineers to motivate security needs and provide indirect design guidance [49].
5. Developers are expected to be aware of various security threats and possible vulnerabilities in their code. This requires the developers to conduct threat analysis and look for possible vulnerabilities, then develop code that refrains the attacks. Overloading developers, who are busy in building software functionalities, with this extra analysis activates will make them the weakest link and make them tend to develop vulnerability-prone code. One of the common practices for developing secure code adopted by developers is to search the web for reusable code to handle an identified vulnerability then copy and paste that code into their own program [18]. This behavior has been shown to often lead to operational but insecure code [18]. Several researchers have proposed some tools and guidance to help developers completing this task properly, see for example [10], [16], [18], [20], [51], [52]. Software architect usually receives vague, incomplete and missed NFR requirements from requirement engineers and hence, consider themselves the real expert to define the NFR [53]. Accordingly, we believe that developers should implement the detailed requirements delivered to them in the design document that should encompass the detailed security requirements as part of the NFR requirements. Keep in mind that when it comes to secure software development, we may think of discussing security design issues even before conducting the requirement engineering phase [54]. This issue needs more investigation and future research.
6. Most of the research done in the security software development view the process in a linear, top-down, waterfall style. This approach seems to be ineffective for many types of projects and is more ineffective when it comes to developing secure software, [55], [56]. Software process engineers should think of new models or extend existing agile concepts to cope with security requirements, see for example [37].

4 Handling Security in the Requirements Engineering Phase

Introducing security-awareness programs for the development team and stakeholders who will specify requirements is not proved to be effective in specifying security requirements. This is because the key players in breaking the security, e.g. adversaries/hackers, are unavailable during the development process. Researchers and practitioners should be careful when introducing new practices or milestones to the requirements process in order not to ruin the whole development process and make it delayed more.

The question is, are the users and customers interested in knowing the details of how their system can be abused/misused and how to protect it? Users and customers in the requirements engineering phase will raise their security concerns at a high level of abstraction, usually at the system level, without bothering themselves in how to implement them. A good requirement engineer can follow the guidelines presented in [22] to map the system's NFR to software FUR, see Figure 1. These software FURs are still high-level requirements and can be used by the designer/architect to develop the detailed technical requirements of the proposed solution. These high-level system NFRs and their corresponding FUR represent the security goals in Turpe's model of security needs dimensions [49], see Figure 2.

This is aligned with our discussion in this regard where we suggest that in the requirement engineering phase, stakeholder should state their security goals as NFR high-level requirements. Involving stakeholders in security-awareness programs, writing abuse/misuse cases or discussing design issues will result in naïve security requirements elicitation process that may end up with incomplete and unrealistic requirements which will be reworked by the designer/architect, hence increase the security debt.

5 Handling Security in the Design Phase

Specifying security needs is not a straightforward process rather it is hard to achieve. While much effort is invested to integrate security and software engineering activities, there has been little work describing how design techniques can be applied to designing secure systems [54].

Designing secure software is a three-variable problem, as shown in Figure 3 [49], security goals, security design, and threats. Any change in one dimension may entail changes or new questions in the remaining two [49]. The intersection between every two variables generates an analysis task, see Figure 3. For more details, you are encouraged to read [49]. Note that the risk analysis can be specified via the what-if scenarios during the requirements engineering phase as the effect of technology is minor. CORAS [57], [58] can be used to conduct the risk analysis. Security design analysis looks for possible unmitigated vulnerabilities that allow

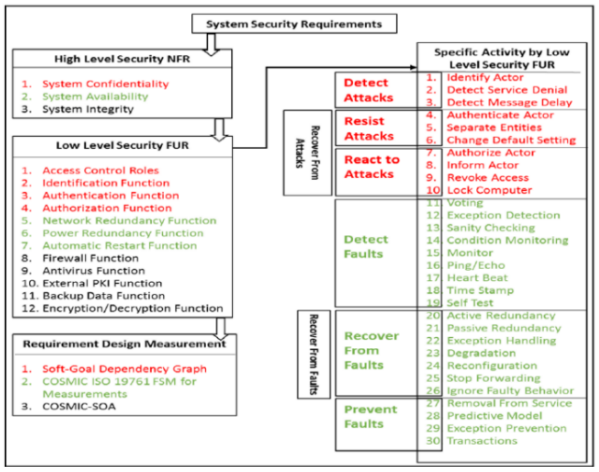

Figure 1: System and software security requirements [22]

attacks to succeed [49]. As the security design analysis depends on adopted technology, it should be done during the design phase, specifically architectural design. STRIDE threat analysis model and Data-Flow-Diagrams DFD can be used to conduct this analysis. CAPEC model [59] can also be used in this regard.

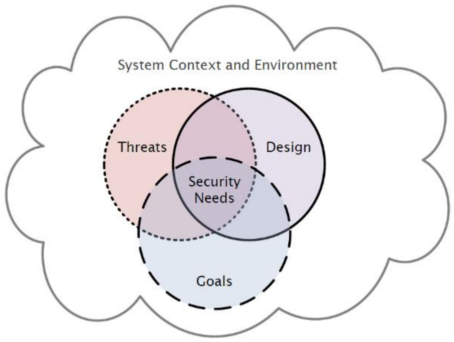

Fig. 2 Dimensions of security needs [49]

Moreover, the design process of a system translates security goals into design choices. This model is found to be the most appropriate model documented in the literature that describes the core activities in the design phase. Note that design process to develop the design and security design analysis to identify possible threats are done in the design phase while the risk analysis and goals identification are more suitable for the requirement engineering phase where the security requirements are collected as high-level non-functional requirements [22]. This model needs more work to specify in detail the practices to follow in each part of it. Wrapping up, we suggest that requirements engineers and developers not to mix practices among the two phases in a way that put the cart before the horse. Such behavior will make the specified requirements vague, incomplete and will give hard time for designers to do their tasks. Table 1 illustrates the main security-related tasks in both the requirement and design phases of the development process.

6 Limitations

As mentioned in the literature review section, developing security conscious software to face the increasing amount and quality of cyber-attacks is still lacking maturity and the research in this domain is still in its infancy phase. Such immaturity forced us to conduct an ordinary literature review. We think a more rigorous literature review is needed in the future. Normally, conducting an ordinary literature review affects the validity of the conducted research but in this research, and due to the relative novelty of the discussed topic, a systematic literature review is unachievable.

7 Conclusion and Future Work

Various researchers and practitioners are mixing some practices and deliverables up when they specify the security requirements during the analysis and design phases. This makes secure software development vague and introduces flaws in the development process itself. The discussions presented in this research shed light on what has been achieved when talking about secure software analysis and design and what problems and mistakes both researchers and practitioners have committed. We suggested some corrective actions to fix these mixed-up activities and the proposed flow of practices in the two phases that conform to the basics of the requirement and design concepts. The suggested modifications should amend the currently available secure software processes. We believe that further research is still needed to improve the software analysis and design phases aiming to make the final software less vulnerability prone. Starting the secure development process with design rather than requirement then iterate between the two phases, accordingly, needs more investigation as well.

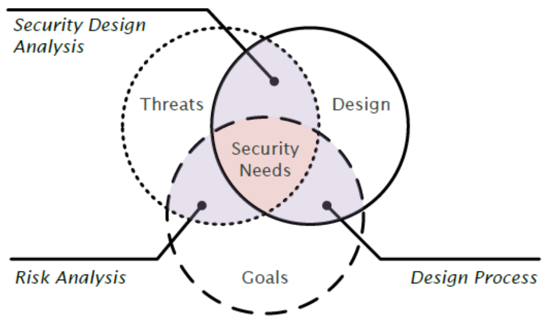

Fig. 3 Analysis tasks. Each pair of dimensions leads to a distinct perspective [49].

Table 1: Tasks to build the security requirement into the system

Phase	Task	Comments and Tools
Requirement	Identify Security Goals	Use the systems and software security requirement framework as depicted in [22] Use the what-if scenarios for risk analyses when developing normal use-cases. Use CORAS for Risk analysis
	Translate security goals into non-functional security req.	
	Mapp NFR security req. into functional user req.	
	Conduct Risk Analysis	
Design	Design the system taking the security FUR into consideration	Develop secure architecture Develop secure detailed design
	Security Design Analysis	STRIDE and DFD

REFERENCES

[1] B. Russell, "IoT Cyber Security," in *In Intelligent Internet of Things*, 2020, pp. 473–512.
[2] R. L. Nord, Ipek Ozkaya, and Forrest Shull., "Software vulnerabilities, defects, and design flaws: A technical debt perspective," in *In Fourteenth Annual Acquisition Research Symposium*, 2017.
[3] M. Linares-Vásquez, Gabriele Bavota, and Camilo Escobar-Velásquez, "An

empirical study on android-related vulnerabilities," in *IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, 2017, pp. 2–13.

[4] A. Takanen, J. D. Demott, C. Miller, and A. Kettunen, *Fuzzing for software security testing and quality assurance*. Artech House, 2018.

[5] Microsoft, "Security Development Lifecycle." [Online]. Available: https://www.microsoft.com/en-us/securityengineering/sdl/. [Accessed: 10-Aug-2019].

[6] G. McGraw, S. Migues, and B. Chess, "The Building Security In Maturity Model (BSIMM)," 2009. [Online]. Available: https://www.bsimm.com/. [Accessed: 12-Aug-2019].

[7] ISO/IEC JTC 1/SC 27 Technical Committee, "ISO/IEC 27034-1:2011 - Information technology -- Security techniques -- Application security -- Part 1: Overview and concepts," 2011.

[8] J. Smith, B. Johnson, E. Murphy-Hill, B.-T. Chu, and H. Richter, "How Developers Diagnose Potential Security Vulnerabilities with a Static Analysis Tool," *IEEE Trans. Softw. Eng.*, pp. 1–21, 2018.

[9] M. Backes, K. Rieck, M. Skoruppa, B. Stock, and F. Yamaguchi, "Efficient and Flexible Discovery of PHP Application Vulnerabilities," in *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*, 2017, pp. 334–349.

[10] M. Backes, K. Rieck, M. Skoruppa, B. Stock, and F. Yamaguchi, "Efficient and Flexible Discovery of PHP Application Vulnerabilities," in *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*, 2017, pp. 334–349.

[11] H. Assal, S. Chiasson, and R. Biddle, "Cesar: Visual representation of source code vulnerabilities," in *2016 IEEE Symposium on Visualization for Cyber Security (VizSec)*, 2016, pp. 1–8.

[12] G. McGraw, "From the ground up: the DIMACS software security workshop," *IEEE Secur. Priv.*, vol. 1, no. 2, pp. 59–66, Mar. 2003.

[13] G. Mcgraw, "Software security," *IEEE Secur. Priv. Mag.*, vol. 2, no. 2, pp. 80–83, Mar. 2004.

[14] H. Assal and S. Chiasson, "Motivations and Amotivations for Software Security," in *SOUPS Workshop on Security Information Workers (WSIW)*, 2018, pp. 1–4.

[15] H. Assal and S. Chiasson, "Think secure from the beginning," in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems - CHI '19*, 2019, pp. 1–13.

[16] M. Green and M. Smith, "Developers are not the enemy!: The need for usable security apis," *IEEE Secur. Priv.* , vol. 14, no. 5, pp. 40–46, 2016.

[17] H. Assal and S. Chiasson, "Security in the software development lifecycle," in *Fourteenth Symposium on Usable Privacy and Security ({SOUPS}*, 2018, pp. 281–296.

[18] Y. Acar, C. Stransky, D. Wermke, C. Weir, M. L. Mazurek, and S. Fahl, "Developers Need Support, Too: A Survey of Security Advice for Software Developers," in *2017 IEEE Cybersecurity Development (SecDev)*, 2017, pp. 22–26.

[19] Y. Acar *et al.*, "Comparing the Usability of Cryptographic APIs," in *2017 IEEE Symposium on Security and Privacy (SP)*, 2017, pp. 154–171.

[20] G. Wurster and P. C. van Oorschot, "The developer is the enemy," in *Proceedings of the 2008 workshop on New security paradigms - NSPW '08*, 2008, pp. 89–97.

[21] Jeremy Dick, Elizabeth Hull, and Ken Jackson, *Requirements Engineering*, 4th Edition. Springer, 2017.

[22] K. Meridji, K. Al-Sarayreh, A. Abran, and S. Trudel, "System security requirements: A framework for early identification, specification and measurement of related software requirements," *Comput. Stand. Interfaces*, vol. 66, p. 103346, Oct. 2019.

[23] O. Tettero, D. J. Out, H. M. Franken, and J. Schot, "Information security embedded in the design of telematics systems," *Comput. Secur.*, vol. 16, no. 2, pp. 145–164, Jan. 1997.

[24] J. McDermott and C. Fox, "Using abuse case models for security requirements analysis," in *Proceedings 15th Annual Computer Security Applications Conference (ACSAC'99)*, 1999, pp. 55–64.

[25] X. Yuan, E. Nuakoh, I. Williams, H. Y.- JSw, and undefined 2015, "Developing Abuse Cases Based on Threat Modeling and Attack Patterns.," *J. Softw.*, vol. 10, no. 4, pp. 491–498, 2015.

[26] T. Srivatanakul, J. A. Clark, and F. Polack, "Effective Security Requirements Analysis: HAZOP and Use Cases," in *International Conference on Information Security*, 2004, pp. 416–427.

[27] I. Alexander, "Misuse cases: use cases with hostile intent," *IEEE Softw.*, vol. 20, no. 1, pp. 58–66, Jan. 2003.

[28] G. Sindre and A. L. Opdahl, "Eliciting security requirements with misuse cases," *Requir. Eng.*, vol. 10, no. 1, pp. 34–44, Jan. 2005.

[29] P. X. Mai, A. Goknil, L. K. Shar, F. Pastore, L. C. Briand, and S. Shaame, "Modeling Security and Privacy Requirements: a Use Case-Driven Approach," *Inf. Softw. Technol.*, vol. 100, pp. 165–182, Aug. 2018.

[30] G. Sindre, "Mal-Activity Diagrams for Capturing Attacks on Business Processes," in *International Working Conference on Requirements Engineering: Foundation for Software Quality*, 2007, pp. 355–366.

[31] C. Schmitt and Liggesmeyer P, "A Model for Structuring and Reusing Security Requirements Sources and Security Requirements.," in *REFSQ Workshops*, 2015, pp. 34–43.

[32] K. Fletcher and X. Liu, "Security requirements analysis, specification, prioritization and policy development in cyber-physical systems," in *Fifth International Conference on Secure Software Integration and Reliability Improvement-Companion*, 2011, pp. 106–113.

[33] Q. Feng, R. Kazman, Y. Cai, R. Mo, and L. Xiao, "Towards an Architecture-Centric Approach to Security Analysis," in *13th Working IEEE/IFIP Conference on Software Architecture (WICSA)*, 2016, pp. 221–230.

[34] Z. Ahmad, M. Asif, M. Shahid, and A. Rauf, "Implementation of Secure Software Design and their impact on Application," *Int. J. Comput. Appl.*, vol. 120, no. 10, 2015.

[35] M. T. J. Ansari, D. Pandey, and M. Alenezi, "STORE: Security Threat Oriented Requirements Engineering Methodology," *J. King Saud Univ. - Comput. Inf. Sci.*, p. In Press., Dec. 2018.

[36] C. Dougherty, K. Sayre, R. Seacord, and D. Svoboda, "Secure design patterns (No. CMU/SEI-2009-TR-010)," 2009.

[37] H. El-Hadary, S. E.-K.-J. of advanced research, and undefined 2014, "Capturing security requirements for software systems," *J. Adv. Res.*, vol. 5, no. 4, pp. 463–472, 2014.

[38] S. Faily, "Engaging stakeholders during late stage security design with assumption personas," *Inf. Comput. Secur.*, vol. 23, no. 4, pp. 435–446, Oct. 2015.

[39] E. Fernandez, "A Methodology for Secure Software Design.," in *Software Engineering Research and Practice*, 2004, pp. 130–136.

[40] Ş. Şentürk, H. Yaşar, and İ. Soğukpınar, "Model Driven Security in a Mobile Banking Application Context," in *Proceedings of the 14th International Conference on Availability, Reliability and Security - ARES '19*, 2019, pp. 1–7.

[41] J. Jürjens, *Secure Systems Development with UML*. Springer, 2005.

[42] J. Jürjens, "UMLsec: Extending UML for Secure Systems Development," in *International Conference on the Unified Modeling Language*, 2002, pp. 412–425.

[43] T. Koch, "Towards Scenario-Based Security Requirements Engineering for Cyber-Physical Systems," in *International Conferences on Software Technologies: Applications and Foundations*, 2018, pp. 633–643.

[44] J. Jürjens and P. Shabalin, "Tools for secure systems development with UML," *Int. J. Softw. Tools Technol. Transf.*, vol. 9, no. 5–6, pp. 527–544, Oct. 2007.

[45] M. Shin and H. Gomaa, "Software requirements and architecture modeling for evolving non-secure applications into secure applications," *Sci. Comput. Program.*, vol. 66, no. 1, pp. 60–70, Apr. 2007.

[46] K. Lano, "Design Patterns: Applications and Open Issues," in *Cyberpatterns*, 2014, pp. 37–45.

[47] Theodor Richardson and C. Thies, *Secure Software Design*, 1 edition. Jones & Bartlett Learning, 2013.

[48] K. Rindell, K. Bernsmed, and M. G. Jaatun, "Managing Security in Software," in *Proceedings of the 14th International Conference on Availability, Reliability and Security - ARES '19*, 2019, pp. 1–8.

[49] S. Türpe, "The trouble with security requirements," in *25th International Requirements Engineering Conference (RE)*, 2017, pp. 122–133.

[50] J. Geismann, C. Gerking, and E. Bodden, "Towards ensuring security by design in cyber-physical systems engineering processes," in *International Conference on Software and System Process*, 2018, pp. 123–127.

[51] Jing Xie, H. R. Lipford, and Bill Chu, "Why do programmers make security errors?," in *2011 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 2011, pp. 161–164.

[52] M. Yoshizawa, H. Washizaki, Y. Fukazawa, T. Okubo, H. Kaiya, and N. Yoshioka, "Implementation Support of Security Design Patterns Using Test Templates," *Information*, vol. 7, no. 2, p. 34, Jun. 2016.

[53] D. Ameller, C. Ayala, J. Cabot, and X. Franch, "Non-functional Requirements in Architectural Decision Making," *IEEE Softw.*, vol. 30, no. 2, pp. 61–67, Mar. 2013.

[54] S. Faily, "Why Designing for Usability and Security is Hard," in *Designing Usable and Secure Software with IRIS and CAIRIS*, Cham: Springer International Publishing, 2018, pp. 3–8.

[55] B. Nuseibeh, "Weaving together requirements and architectures," *Computer (Long. Beach. Calif).*, vol. 34, no. 3, pp. 115–119, Mar. 2001.

[56] T. Heyman, K. Yskout, R. Scandariato, H. Schmidt, and Y. Yu, "The Security Twin Peaks," in *International Symposium on Engineering Secure Software and Systems*, 2011, pp. 167–180.

[57] B. Solhaug and K. Stølen, "The CORAS Language – why it is designed the way it is," in *11th International Conference on Structural Safety and Reliability (ICOSSAR'13)*, 2013, pp. 3155–3162.

[58] M. S. Lund, B. Solhaug, and K. Stølen, *Model-Driven Risk Analysis: The CORAS Approach*. Oslo, Norway: Springer, 2011.

[59] "Common attack pattern enumeration and classification (CAPEC)." [Online]. Available: http://capec.mitre.org/. [Accessed: 18-Aug-2019].