Safety Risks and Security Threats in Low-code Software Development

Alkisti Onoufriou
a.onoufriou@student.utwente.nl
University of Twente
Enschede, The Netherlands

ABSTRACT

The demand for application development is increasing exponentially, however there is a lack of IT talent to meet the market demands. Organizations are switching to low-code development in order to address this issue. Low-code development enables businesses to meet their needs outside their IT department, but there are certain safety and security implications involved. Research can help identify these implications, in an effort to support citizen developers in building safe and secure applications. For this purpose, this work aims to 1) identify which safety risks and security threats involved in software development apply to low-code development practices, 2) jointly model these risks and threats through the use of a multi-level model and a scenario, 3) provide guidelines reflecting the identified points of awareness for developers.

KEYWORDS

low-code development · safety · security · case study · multi-level model

1 INTRODUCTION

In today's fast-developing technological society, corporations are put in a position in which they have to keep up with the tools, demands and requirements of a constantly evolving industry. This need for minimum time-to-market and more human resources when building software solutions, gave low-code development platforms the opportunity to shine and start becoming an integral part of many enterprises' IT projects and overall workflows [1]. Low-code development is a visual approach to software development, that enables faster delivery of applications through minimal hard coding [2]. The end users of these platforms are among others business analysts, marketing professionals and to a large extent citizen developers. Citizen developers are defined as users with little to no coding experience who build applications with IT-approved technology [3].

In all software development platforms there are certain safety risks and security threats involved in the development process and also in the resulting applications. When it comes to low-code, this is not adequately explored in existing scientific literature. The distinction and interactions between safety and security are also still ill-understood in the literature, adding value to this work [4], [5]. The two concepts entail different aspects of a system and are highly interrelated, making it important to look at them jointly, in order to properly and holistically assess the risks and threats of a system and be able to model them in a complete way [6].

The topic of low-code development is more relevant than ever, since the use of low-code development platforms by citizen developers is expected to rise in the upcoming years. Gartner predicts that by 2026, developers outside formal IT departments (citizen developers) will account for at least 80% of the user base for low-code development tools, up from 60% in 2021 [7]. In addition to this, a clear gap in the existing literature has been identified when it comes to the risks and threats of using such platforms. This gap could be related to the fact that it is a relatively new technology ([8]), but nonetheless it is something that especially IT experts slowly start to highlight (89% of IT leaders that talked with SalesForce in 2017 focused on the security issues of low-code development among other things [9]). This work aims to bridge this gap and produce new knowledge by looking into the risks and threats in software development, adjusting them to the low-code context while adding value by the use of the multi-level model and a safety-security categorization. A combination of methods is used to achieve these outcomes, including a case study, qualitative analysis and modeling.

This document is structured as follows: in the following subsection (1.1) the research question and sub-research questions are defined, followed by section 2 where the related work is presented. Section 3 outlines the methodology for answering the defined research question. In section 4 the results of the conducted case study are presented and discussed, followed by section 5 where these results are modeled and put into a safety and security categorization. Section 6 is concerned with validating all the presented work which is followed by section 7 that provides a critical reflection and discussion of all results. All this leads to sub-section 7.3 that presents the proposed guidelines for developers, as the final outcome of this work. The document is concluded with section 8, outlining the contribution of this work to the current body of knowledge.

1.1 Research Question

In this research project the following research question will be investigated:

> RQ: What are the main safety risks and security threats involved in low-code software development and how can they be modeled to best provide insights for developers?

The research question aims to fill a literature gap by analyzing and modeling risks and threats in low-code development, starting from traditional development as a baseline. Naturally, some sub-research questions arise from the main research question that roughly outline the steps and knowledge gaps to be filled, in order to answer the main research question.

> Sub-RQs:
> 1) What are the main safety risks and security threats in software development?
> 2) Which of these risks and threats apply to low-code development and why?

3) Which of these risks and threats are related to safety and which to security?
4) How can they be showcased in a model and through a scenario?

2 RELATED WORK

This section covers the related work (i.e. literature) in both the safety and security domain and the domain of low-code software development. Due to the nature of the research, preliminary literature review had to be performed in the two domains separately, in order to gather an in depth understanding of the existing work, before advancing to the review of literature in the risks of software development. As mentioned, the field of low-code development is relatively new, hence there is limited scientific literature in the domain.

When it comes to safety and security, there exists literature that aims to define these concepts and highlight the importance of considering both aspects together in software development.

[6] have proposed a complete development cycle combining the execution of safety and security activities simultaneously, by integrating both, in each phase of the cycle. Their work was evaluated through two hypothetical case studies in the field of cyber physical systems, showcasing the importance of both safety and security in the development lifecycle.

[5] conducted a literature review aiming to define and conceptualize the concepts of safety and security focusing on the interpretive issues, and their complex interactions that could result in risk amplification, unintended consequences and 'blind spots' in a developed system. This is particularly useful for this work, since it helps define and use these concepts in practice.

Lastly, when it comes to modeling safety and security, [4], and [10] and [11] have conducted an analysis showcasing the gaps of existing literature in the domains, a multi level analysis of the roots of the interactions between the two concepts, and the nature of their relationship by means of a use case in the area of automotive systems, respectively. It should be noted that there is also literature covering more aspects of model-based safety and security that have been reviewed (e.g. [12], [13]), with a focus on their overlaps.

In the domain of low-code software development, there is a clear gap in the literature when it comes to its safety and security implications. Nonetheless, some important work is presented to help understand the process of low-code development, which can help in identifying applicable safety risks or security threats from traditional development. In addition, for the purpose of this research, certain platforms' documentation and research institutes in the field are also of value (e.g. [14], [15]).

[16] conducted a literature review focused on the challenges in low-code software development and corresponding mitigation suggestions. Concepts like citizen development and the architecture of low-code development platforms (LCDPs) are related to the identified challenges and some of these challenges could be addressed in the concept of security, making the work highly relevant for this research.

A significant find was also the work by [8] on the vulnerabilities in LCDPs, highlighting the lack of knowledge regarding underlying vulnerabilities of developing and deploying low-code applications. Their research is focused on the vulnerabilities of the apps that have been developed using low-code technologies. The outcome of their research is a list of the top three vulnerabilities in three defined perspectives, which were derived by combining different data sources. These vulnerabilities can also be directly or indirectly linked to security aspects of the development process.

This review served as a starting point towards establishing a background to all the concepts involved in order to meet the research objectives. The next phase was focused on literature highlighting safety threats and security risks in software development as a whole, in order to apply this knowledge to the context of low-code development. The outcomes of this phase were used to reach the findings of this research, as further explained in section 3.

In this regard, [17], [18], [19], [20] discuss important threats involved in all types of software development projects, shedding light on the main aspects involved.

[21], [22] and [23]'s work focuses on the risks and vulnerabilities in specifically the development and deployment of web applications. This is of utmost relevance for this research since low-code platforms are primarily used for web application creation. [24] and [25] elaborate on the dangers and issues of agile software development, which is another characteristic of low-code development. [26]'s work is also of great value, since it is a mapping study and survey covering all risks involved in the use of Off-The-Shelf components. His work thoroughly analyses the important pitfalls and risk sources of externally furnished and ready-made components. Lastly, [27]'s work focuses on security challenges in software development that makes use of high levels of abstraction. This was included in the review due to the abstract and multi-layered nature of LCDPs.

3 METHODOLOGY

This section describes the steps followed in order to answer the chosen research question.

Firstly, a preliminary literature review was carried out in order to identify and evaluate the relevant literature in the field, and obtain a complete overview of the areas being studied [28].

Then, in order to answer the first sub-research question a case study was conducted. A case study is the systematic investigation of a specific area or unit, used for the comprehensive exploration of research questions [29]. In this case, it refers to the comprehensive review of literature on risks and threats in software development, and the synthesis of findings from all reviewed studies. Low code and high code development can be seen as separate units within the area of software development. This methodology also enables the comparison of similarities and differences of such units ([29]), making it useful for this work.

The focus was on vulnerabilities and threats in software development and their potential categorizations. The results were obtained from 9 scientific papers, including 5 systematic literature reviews ([21], [24], [17], [19], [16]). Databases used were: Google Scholar, IEEE Xplore Digital Library, FINDUT and the Science Direct digital library due to their comprehensive collections of papers and accessibility. Search keywords included risks, vulnerabilities, threats, software development, web application development, and low-code development to ensure a broad yet relevant search scope.

The papers chosen for review cover multiple areas of software development namely traditional, agile and development including the use of Commercial Off The Shelf (COTS) components and high levels of abstraction. Since the ultimate aim of this research was to identify the risk factors involved in low-code software development, the aforementioned sub-categories of traditional software development are relevant, because they entail characteristics that are present in low-code development.

Additionally, concerning the first sub-research question, the aim is to create a list of risks and threats in software development, based on the reviewed literature. After rigorous review of the selected papers, the outcomes of the papers were summarized in an initial table of risks and risk factors that consisted of 219 entries. These entries were then divided into two categories depending on whether they are inherently of non-technical or technical nature. This grouping provided a clear segmentation, organizing the results and making their management and analysis simpler. This also allowed for more focused discussion and reporting, and enabled the identification of relationships between factors. The next step was an initial filtering of the obtained results, that comprised the removal of duplicates and merging identical factors that were phrased in different ways within the literature. Next, the factors that did not concern the developers (i.e. relating to managerial or other issues) and factors that were too generic, in the sense that they were true for any project and not just software development projects, were omitted. This helped to narrow down the results, focusing on this work's objectives. Further analysis and additional research followed, in order to pinpoint which of the identified risks and threats are most relevant for low-code development and thus answering the second sub-research question. Finally, the results were qualitatively analysed and classified with respect to which phase of the Software Development Life Cycle (SDLC) they occur in. The final results are presented in a simple, clear and concise manner in Table 1 and 2.

The next step was modeling the identified risk factors in the multi-level model proposed in [10], based on the existing analysis. Alongside this model, in order to answer the third sub-research question, all factors were categorized in terms of safety and security. This analysis is discussed in section 5. In addition to this, the same model is used to model the risk factors involved in a chosen, real-life scenario. The scenario was chosen due to its impact in the field of low-code and its nature, as it demonstrates the multi-level model's capabilities and allows the discussion of certain important risk factors in more detail. This modeling answers the fourth and final sub-research question.

The next phase addresses the validation of the research outcomes. For this purpose, two industry experts from the field of security and low-code development were interviewed. In this way, insights, attitudes and opinions regarding the research results were gathered and extracted, allowing the critical assessment of the validity of the research findings. The research findings combined with the gathered input, enabled the formation of guidelines for developers in low-code software development. Upon completion of this stage, the research is concluded and all research results are obtained.

4 CASE STUDY

This section provides a thorough discussion of the case study results.

The results can be found in Table 1 and 2 in the Appendix (9). They are presented in a tabular view with respect to which phase of the SDLC they can occur in and the phases considered are: requirement engineering, design, coding/development, testing, deployment, maintenance. The categorization of the factors with respect to these phases was presented by [17], and is useful to structure and group the findings in a clear way, hence it is used in this study as well. For this research an extra category was added, the entire SDLC category, which refers to the factors that can take place throughout all the defined phases.

As mentioned in section 3, in order to simplify the qualitative analysis, the factors were divided into inherently *non-technical* and *technical* in the first phase of the study.

The *non-technical factors* pertain to human and procedural aspects of the development process, focusing on the essential work happening around feature implementation, rather than the implementation itself. These factors still impact the developer's ability to effectively use the low-code platform. For example, in this category factors entailing planning, requirement engineering, skill/knowledge gaps and training are included.

The identified *technical factors* in low-code development, focus on the technical aspects of using and working with the low-code platform. These can relate to integration, scalability, functionality limitations and other technical matters. It must be noted that potential attacks against the resulting system/application are also included in this category. In addition, these potential attacks are classified under the Coding phase of the SDLC since their roots usually relate to gaps in code or configurations.

Both *non-technical* and *technical* factors are identified for all phases of the SDLC, except the Requirements Engineering phase were only non-technical factors were identified, and the Testing and Deployment phases were only technical factors were identified. The following sub-sections contain separate analyses of all phases for both categories and a discussion of certain points of interest.

4.1 Non-technical factors

This sub-section contains an analysis of the key points of the identified non-technical risk factors, with respect to the SDLC phase they occur in.

i. Requirements Engineering Phase

The first phase of the SDLC is the Requirements Engineering phase. It deals with the definition of objectives and requirements for the project, answering questions such as: What will be developed and why? What features will be included in the final product? and others. This phase continues to be recognized as the key to on-time and on-budget delivery of software projects ([30]), hence it is vital for the considerations of security and corresponding safety risks, to start in this phase.

The main source of risk in low-code development identified for this phase is unclear objectives or requirements and the negligence of non-functional requirements, including security. In low-code platforms, with the focus being on Rapid Application Development (RAD), the emphasis on clear requirements might be overlooked

in the face of 'jumping' straight to development [18]. This can lead to compromised quality and security of the resulting service. In addition, more often than not, applications built in low-code platforms are developed by business professionals, in order to meet their business needs [31]. This might lead to a less formalized requirements engineering process, that can then result to technical debt. This also relates to the lack of COTS-driven requirements engineering process to be followed, mentioned in [26].

The negligence of security requirements is also a significant risk factor in this phase, and can be the result of the lack of awareness of their role and importance by the citizen developers. It is vital for security requirements to be considered in all phases starting from their definition (in this phase), followed by their careful implementation, evaluation and management [17]. Overall, the requirements engineering phase is a highly cited factor relating to the development of secure software [17]. This remains the case in the area of low-code development.

ii. Design Phase

The next phase is the design phase which is concerned with the way and means used to achieve the defined objectives and requirements.

In low–code development, this entails platform selection, planning and risk assessment of the selected platform and resulting application. Regarding the selection process, the developers need to evaluate all aspects of the candidate platforms in terms of matching with the specified requirements and security. Failure to do so could result in a vulnerable or even useless system. Two interesting findings in this phase are the overly optimistic learning curve ([26]) and inadequate assumptions ([24]). Citizen developers should be aware that low-code platforms still require familiarization with a user interface and the technicalities involved, in order to properly build an app.

Low-code platforms make use of multiple levels of abstraction, and security issues arise when implementations diverge from this abstract intuition [27]. Thus, in order for developers to defend their system against threats, they need to be aware of all the different layers that are 'abstracted away' from them. It is also important for citizen developers, just like all application developers, to be aware of potential vulnerabilities and their consequences prior to software development [23]. The first step in this direction is thorough risk assessments during this phase.

Alongside their abstract nature, low-code platforms are also characterized by building on Model-Driven Software Development [32]. This brings another dimension of risks in the discussion, that goes further than the identified factor of the lack of abuse case models, attack patterns and data flow diagrams. This dimension includes the risks of creating/generating wrong models, which in turn can highly affect the safety of the resulting system. There are a lot of aspects related to poor quality models and in part they relate to the modeler's lack of knowledge [33]. This means that citizen developer's may lack the modelling knowledge, the modelling language knowledge, which relate to semantic and syntactic aspects of a chosen modelling language, or the domain knowledge. This leads to the creation of incomplete models or models not representative of the underlying, organizational or other, real-world structure.

iii. Coding Phase

For this phase, the only non-technical factor identified in the literature is the lack of difference between developer's role and security reviewer to have objective results, presented in [17]. In low-code development, citizen developers are the ones to develop the application and configure, implement and test its security settings. This can lead to subjective results or even dropping security testing as a whole, in favor of releasing the software as quickly as possible [25].

iv. Maintenance Phase

In this phase, the web application has been developed through the selected LCDP, and it is released and used within the workflow. Risk factors connected to this phase are somewhat outside the control of the individual developers and have to do with the low-code technology itself.

An important risk factor here is vendor lock-in. Vendor lock-in, refers to a situation in which a client is dependent on a certain provider in order to complete their computing needs, as stated in [34]. This leads to the customer being highly dependent on the platform provider for all aspects of their own systems and services, including security management and defense, which could result in dangerous mismatches of vulnerabilities.

Similarly, as mentioned in [26], the developers have reduced control of the future evolution of their system, since it follows the steps of the platform provider. This risks the system becoming obsolete or posing difficulties in upgrading an already developed and in-use system.

v. Entire SDLC

Naturally, there are certain factors that concern the entirety of the project's life cycle and are inherent to low-code software development. These can be seen in the last row of Table 1. An important factor has to do with the lack of comprehensive documentation from the platform providers ([16]), but also by the developers themselves. In low-code development, the few lines of code that are written do not have comments and do not always follow the field's best practices ([35]), which can create additional vulnerabilities due to the lack of structure and consistency. Overall, documentation is one of the aspects that can be easily overshadowed by speedy delivery in low-code environments.

Another thing that developers need to keep in mind is the credibility of the vendors [26]. The market of low-code development is expected to be worth approximately 137 billion euros by 2030, meaning it is becoming increasingly competitive [36]. With this in mind, each vendor tries to become as appealing as possible to all potential customers. This means that quality expectations need to remain in check, and no claim should be treated as a fact, to prevent exploitable loopholes in released systems.

4.2 Technical Factors

This sub-section contains an analysis of the key points of the identified technical risk factors, with respect to the SDLC phase they occur in.

i. Design Phase

Similarly to the non-technical factors, the technical factors identified in the Design phase, have to do with the selection of a LCDP.

As mentioned, the market of low-code development is competitive and expanding [36]. Nowadays, enterprises do not only have to choose whether to include low-code development in their workflow but if they do, which platform to go with. As shown in the findings, shortfalls in externally furnished components and unknown features and quality, can be sources for vulnerabilities and also complicate the identification of the root of occurring defects for the developer. This is something directly linked with platform selection, as it depends on the components each vendor provides.

ii. Coding Phase

The coding phase is where applications are actually developed by the (citizen) developers, so security of the system becomes concrete by the implementation of access rules, authentication, data security and other similar features.

Two interesting findings were found in [26]'s work regarding Off-The-Shelf based software development, and relate to the upgrade to new versions of the used components. This is extremely relevant for low-code development since there are constant releases of new versions of the development platforms, and these could take place post-release, or even mid-development of an application. This makes it difficult for developers to keep up with the platform and can lead to increased complexity, while the unneeded and unused services may interfere with intended functionality, posing serious risks to the system [18].

Security (or other) misconfigurations and accessibility issues are also crucial for building secure low-code applications and can be challenging for developers [16]. Moreover, data security is also a root of common threats like information leakage and data exposure in low-code development [16]. [1]'s survey found that data security was mentioned as the top concern of IT professionals regarding citizen development. Regardless of the numerous out-of-the-box data storing options provided by low-code platforms, the security and access rules of the stored information is up to the developer to dictate and configure. If this does not happen in a rigorous manner, it can be detrimental not only for the system or user, but for the whole organization.

In this part it is also important to briefly discuss the potential attacks (as defined by [17]: An actual event done by a person, in order to harm as an asset the software, through exploiting a vulnerability) that could take place in a web application developed in a low-code environment. In the market of low-code platforms there are vendors that claim that it is impossible for injection attacks to take place when developers use their native components [37]. Even though it is not in the scope of this research to test that claim, interestingly, common web application vulnerabilities are cited in literature in the context of low-code applications [8]. Specifically, injection attacks like cross-site request forgery, cross-site scripting or SQL injection are vulnerabilities that can result from the use of low-code development platforms [8]. Moreover, alongside this type of attacks, there are other attacks like brute force and password spraying that have been observed in leading low-code platforms in the past five years [38]. This also highlights the risk involved in vendor claims and the importance for their critical assessment.

iii. Testing Phase

In the testing phase the developed software is evaluated with respect to its features, functionality and security. All aspects of the system need to be considered, in order for existing vulnerabilities and errors to be identified and resolved before deployment.

In low-code development the main type of testing that takes place is so-called black box testing. Black box testing is a testing technique in which the internal workings of the software are not known to the tester. The tester only focuses on the input and output of the software [39]. While this technique does allow the comprehensive evaluation of the features and functionality of the software, the hidden layers of the system are not covered by the test cases. Therefore, the security of the overall system cannot be rigorously evaluated or guaranteed solely based on whether it performs the intended functions. Overall, limited testing is a challenge faced in low-code development ([16]) and it can lead to the deployment of vulnerable systems, leaving room for exploitation.

iv. Deployment Phase

The next phase of the SDLC is concerned with deploying the developed system, which includes its integration to the existing architecture or environment of the enterprise.

Low-code practices accelerate development, foster closer collaboration between software development and business teams, and enable the rapid organizational response to market demands ([40]), increasing project agility. As a result of its frequent and rapid iterations, applications and software can end up having systems deployed that are not fully understood, or do not follow, the security solutions and policies in place [25]. An architectural mismatch or underestimation of integration efforts then takes place. This could also take place due to the lack of technical knowledge of the citizen developers. In turn, this can lead to gaps in the enterprise's architecture and vulnerable access points to the whole organization's environment.

v. Maintenance Phase

This final phase of the SDLC regards the activities taking place post-release of the system. It is focused on continuous improvement, scalability as well as error handling.

As mentioned in [16], in the context of low-code there are scalability considerations that should be addressed in a secure manner to avoid large-scale vulnerable systems.

Insufficient logging and monitoring of threats could also pose certain risks ([21]), and could be the result of unawareness of the users or developers or the lack of the prioritization of security monitoring.

Lastly, configuration management is an important component in the secure maintenance and operation phase of low-code software projects [17]. Serious data breaches have happened in the past in developed applications, due to the use of default configuration ([41]), proving the above statement, and showcasing that vulnerabilities should be accounted for and handled, even after an application is in use.

vi. Entire SDLC

When it comes to the entire SDLC, security needs to be prioritized and accounted for in every step of the way. In the use of low-code platforms, there are factors identified that relate to this.

Firstly, the developers do not have access to the source code behind the functionality of the user interface and the features being used. This significantly limits the knowledge and ability to evaluate the security/defense measures already in place. As mentioned in [27], in security, the defender has to move first, so it is important for defense measures and strategies to be in place and properly understood, before an application is released.

Secondly, low-code applications can be susceptible to high technical debt throughout the entire Life Cycle, where future rework is required. Technical debt is accrued work that is "owed" to an IT system [42]. Sources of technical debt include the negligence of certain requirements, temporary speedy solutions of bugs and errors, or even contracting limitations.

The next part is concerned with showcasing and modeling these potential risks and risk factors.

5 MODELING SAFETY RISKS AND SECURITY THREATS

In this section, a further analysis, categorisation and modeling of the factors identified and discussed above, is presented.

One of the goals of this research is to identify which of the factors refer to safety and which to security, in order to make the analysis of their potential relationships and interactions possible. For this purpose, the factors from the previous section are used as input, and are further analysed to achieve their categorization and modeling.

From the developer's perspective, it is important for these two aspects (safety and security) to be considered together in the development lifecycle ([6]), to ensure the development of both safe and secure systems and applications. This consideration is also critical due to the intertwinement of the two aspects. There are various types of interactions that can take place between them in different situations. For example a system's safety measure might strengthen, hinder or have no effect to the system's security (this is further explained in 5.1) [10]. This makes their joint consideration a necessary step for safe, secure and complete development.

In the field, safety and security take different forms when different definitions are applied and the terms are sometimes even used interchangeably [5]. No precise definitions and meanings are shared by all individuals in the field ([5]), making their distinction unclear. Nonetheless, it remains necessary to investigates both concepts in order to account for all the risks and threats in software development.

In order to adjust the terms to the context of this work, we combine two different approaches/definitions, one presented in [4] and one presented in [5]. In [4], the distinction is clear; safety refers to the absence of unintended failures, and security refers to absence of malicious attacks. This means that safety risks concern accidental failures, for example (on the highest level) a system failure, while security risks concern malicious attacks, and aspects related to intended vulnerability exploitation. In [5], sub-themes for both safety and security are defined, aiming at the holistic representation of the terms, while showcasing the range of aspects they can entail. In essence, security is more focused on all types of potential threats, incident prevention and risk assessment and mitigation, while safety revolves more around all types of potential hazards or accidents. Additionally, it is mentioned that safety science is also concerned with social and psychological factors that incorporate human cognition and behavior and their variability. Combining these two approaches enables the distinction of the identified risk factors in the context of low-code development.

Given the developer's perspective, two safety-security related categorizations of the identified factors are possible. The first one concerns whether the factor is under the developer's control, so whether the developer can change or influence it and the second one concerns whether the developer knows about the factor, so whether the developer is aware of the situation despite having or not having control over it. For the final categorization the latter distinction was chosen. This is because it incorporates the intended versus unintended nature of security and safety respectively, as well as human cognition and behavior when it comes to the developer's awareness of the selected factors. To be clearer, when a developer is unaware of the existence of a factor due to lack of knowledge or due to their own perception and behavior, that factor is categorized as safety related, capturing the accidental and unintended nature of the term. Similarly, security related factors occur when the developer is aware of the situation and it is under the developer's control to properly manage it and mitigate the potential threats. It should be noted that attack-specific factors as well as the potential attacks themselves also concern the security of the system. This distinction is visualized in the multi-level model, and section 5.1 provides a further discussion.

5.1 Multi-level Model

This section showcases and analyzes the initial modeling of all the findings. For this part the multi-level model initially proposed in [10], as well as the extension proposed in [11] is used. This decomposition was chosen because LCDPs are multi-leveled complex systems, so it is crucial to investigate which part of the system each factor can affect, which goes alongside the severity of each factor. In the visualization, the distinction between security and safety related factors is also visible (security is denoted with red and safety with blue color text in the model, the numbers in the black brackets denote the occurrence of the factor in multiple levels) and Mendix is used as a reference platform due the variety of open sources and the fact that it is a leading platform for IT Services [43].

The proposed, extended multi-level model consists of four levels, each capturing a different layer/dimension of the complete, complex system. As described in [11], each level represents the following:

Level 0: Component level; self-contained elements that make up a sub-system or system. In the case of low-code development this refers to all the different features of a platform that are used to make up a certain desired functionality. These components are usually in a ready-to-use form but can also be customized and modified by the developers. A good example is certain buttons that platform offer, with pre-defined functionality, like the native back button in Mendix ([44]).

Level 1: Information level; the information that facilitates interaction within or between sub-systems. In the present case, this

consists of all factors related to data storage, data exchange and validation, essential for the safe and secure use of a web application.

Level 2: Sub-system level; functional decomposition of the system, different sub-systems have dedicated functionalities that must be integrated to make the whole system. In the present case, this refers to all sub-processes that make up the whole application's functionality, for which more than one features/components are involved. For example, this could be a specific microflow or workflow within the app like an input validation flow or others.

Level 3: System level; represents the system as an atomic unit and also includes its high-level objectives. In the case of low-code this relates aspects affecting the whole, developed application and also certain characteristics of the whole LCDP.

Level 4: System level, human; represents the human aspects involved. In [11] this level is limited to the needs or consequences for a human to be safe, but for the sake of this research we extend this to include all aspects related to human behavior, perception and roles involved.

Figure 1 presents the resulted multi-level model. The following analysis contains certain points of interest and the interactions between safety and security in LCDPs. This relationship is analysed within and across the levels, with respect to mutual reinforcement, antagonism and independence. As defined in [10], mutual reinforcement refers to the situation when fulfillment of safety requirements enhances security (or vice versa), antagonism refers to conflicting requirements or effects between safety and security, and independence is the case when there is no interplay between the two.

It is worth noting that the relationship between safety and security within the same level can differ per combination of factors. Dominantly, security reinforces safety within all levels in LCDPs, though independent interactions also occur. Between the levels potential antagonistic interactions are mainly seen. The analysis below explains this further. Assume independence when no between level relationship is observed (or mentioned).

A. Level 0: Component level

In this level, we look at factors related to the lowest level of abstraction within the low-code platform. Individual components or features of the platform are involved, as well as code customizations where business logic threats can occur. Unknown features and quality limitations are important factors here, as all components that can be used, customized or created on each platform, occur on this level.

An interesting observation here is the importance of the identified safety factors. Externally furnished components can be complex and developers are usually unaware of the details behind them, such as the source code ([16]) making their use unsafe without the needed caution. Lastly, there are five factors in this level that belong to two other levels, namely security misconfigurations, lack of proper testing, insufficient threat monitoring, improper risk assessment and poor documentation. Different LCDPs allow security configurations at various levels. For example, Mendix gives the developer the opportunity to configure security both on a global and a module level ([45]), making it relevant for numerous layers of the final system and increasing the points where security misconfigurations could take place. Security risk assessment and analysis are crucial for creating secure software artifacts [17]. With this in mind, assessment and monitoring of security threats, testing and documentation are integral across all three levels (namely the components, the subsystems and the system as a whole) and essential for eliminating vulnerabilities and developing secure applications.

Interactions: When it comes to the relationship of safety and security, it can be seen that addressing security factors, like insufficient testing and improper risk assessment at the component level, enhances its safety by improving reliability and predictability. For example, applying rigorous testing as a security measure, lowers the possibilities of shortfalls in the externally furnished components or logic threats to go undetected, thereby strengthening the component level safety and showcasing a clear reinforcement. A significant relationship here, is how strong component security can affect the safety of the other levels, namely level 1 and 2. Component security mutually reinforces safety in the Information level by securing the stored data with the implementation of the relevant features/configurations and best practices. On the other hand, an antagonistic relationship can be observed with safety on the Subsystem level. Overly complex component security measures, can hinder the safety of a sub-system by making the integration of the different components extremely challenging, potentially causing accessibility issues or unintended failures within workflows, like incorrect log-out implementations.

B. Level 1: Information level

In this level, factors related to information, such as data storage and input validation, are included. Even though the included attacks can affect multiple parts of the system or a sub-system, they revolve around information manipulation or exploitation (Injection-based threats, XSRF, Tampering etc.), hence are included in this level. It is also worth mentioning that insecure data storage and information leakage are categorized as safety issues, since the most common scenario is considered where the developer is unaware of the possibility of wrong-doing, assuming no malicious intent. This can be the result of limited knowledge, awareness or attention to the information related aspects and threats.

Interactions: Within this level, certain security factors (e.g. encryption) can be linked with data storage and information leakage, that when addressed minimize the risk of these potential safety issues. This can also take place vice versa, as the secure data storage can better guard the system against information related security attacks. When it comes to interactions with the safety of the above levels, increased data security can lead to a potentially antagonistic relationship with the safety in levels 2 and 3. This can happen since strict data access governance might highly complicate data flows both within and between sub-systems, which in turn, can lead to availability or workflow execution issues. When it comes to the overall developed system, safety could be compromised, due to performance or reliability issues that could arise from the added

Figure 1: Multi-level model of all identified factors

technical complexity.

C. Level 2: Sub-system level

This level is concerned with activities involving the integration and manipulation of components and features to achieve desired functionality. Notably, most factors at this level pertain to security, which aligns with the nature of these activities.

The incorrect logout implementation is included in this level because it can take place in the sub-system/microflow in which the logout operation is taken care of. Some platforms offer ready-to-use logout functionality ([46]), which makes it a feature of the platform, hence a potential inclusion in level 0, but there is a possibility to implement the signout on the server side using a (micro)flow (also a possibility in Mendix) which can then entail multiple features of the platform and combination of actions, making the categorization into this level possible and a more holistic choice. This factor is categorized into safety, since the developer is usually unaware about the potential of this factor occurring, assuming no malicious intentions from the developer's side.

There are three factors included in both this level and level 3. This is because the system failing to handle errors, can relate to errors occurring both at the sub-system and the system level and also attack and data modeling can and should take place, both for the whole system but also for each implemented functionality.

Interactions: As this level is dominated by security, no significant interactions with safety are identified internally. A significant finding here is the potential antagonistic relationship observed between sub-system security and human-level safety in low-code. In low-code development, usability and user experience are important parts of development, but also of the resulting application. Overly frequent security checks might hinder usability, causing frustration and confusion and leading to unsafe practices or mistakes that can impact the safe use of the developed application.

D. Level 3: System level

As previously mentioned, this level considers the system as an atomic unit, encompassing factors related to the entire system as well as its high-level goals. The requirement engineering process is directly linked to the system's high-level objectives and desired functions, hence all factors relating to this process are included in this level.

An interesting observation here is the categorization of technical debt into safety. There are several potential roots of technical debt within a software development project are often hidden to the developer, making it a safety rather than a security concern. Despite technical debt involving necessary future rework for specific system functionalities, it is addressed at this highest level ([47]), due to the variety of organizational aspects it often entails.

Another interesting categorization is regarding vendor lock-in and the reduced control of future evolution of the system. These are unavoidable factors of low-code development and out of the developer's control. However the developer is aware of them, and can take proactive measures to mitigate the risk they entail, categorizing them as security related factors. Lastly, side channel and interception attacks are included in this level because they have to do with the exploitation of characteristics of the way the system operates, in its whole and final form.

Interactions: Within this level, security mutually reinforces safety for the most part, by leading to increased overall resilience of the system. A system in which the identified security factors are present, will be prone to the corresponding safety risks, such as failure to handle errors and unnecessary complexity. When it comes to the interaction with level 4, the overall safety of a system (i.e. absence of the mentioned safety risk factors), protects against human error and misconceptions, thereby enhancing level 4 security. Conversely, overly strict system-wide security, can affect user-friendliness and users' attitudes, potentially hindering level 4 safety. Additionally, user awareness, assumptions and other aspects mentioned in level 4 are entirely independent of the security or safety measures in place in the system. All of the above contribute to the formation of a relationship between level 3 and 4, that cannot be holistically interpreted by one of the defined interaction types. Rather, the relationship combines reinforcement (level 3 safety with level 4 security), antagonism (level 3 security with level 4 safety) and independence (for specific factors; e.g. faulty vendor claims or inadequate assumptions are not affected by any level 3 measures).

E. Level 4: System level, human

In the fourth and final level, the factors revolve around human behavior and decision making. For low-code development this entails the selection process and the developer's assumptions and perceptions such as overly optimistic learning curve, quality expectations and unrealistic planning. The developers are of course unaware of such behaviors at the time of development making these safety considerations.

Another safety consideration in this level are faulty vendor claims. Developers unawarely taking claims as facts poses serious risk and could result in the development of unsafe and vulnerable systems. Neglected security requirements are included in this human level since it is a factor that depends on the developer's attitude towards these non-functional requirements.

Lastly, even though phishing and scam are types of attacks, they are included in this level because they are based on the exploitation of human perception and the psychology of the system's users.

Interactions: Safety and security are independent within this level due to the diversity and nature of the factors. Independence is also observed with respect to the other levels, as the influence of factors at this level on the system is indirect, complex, and involves numerous external aspects beyond the scope of this research.

All in all, it can be concluded that LCDPs are complex systems that need thorough understanding of all their levels in order to be used safely and securely. It can also be concluded that security and safety can interact antagonistically in certain cases, when it comes to between level interactions, while within the levels, security measures are either independent or mutually reinforcing one another. It is key for the developers to consider these concepts together and aim to find the right balance between safety and security for the system being developed.

5.2 Model-based Scenario Analysis

This section demonstrates the multi-level model's application through a real-life scenario involving the Microsoft PowerApps LCDP and a number of the identified factors. It emphasizes the importance of developer awareness and caution in LCDP usage, aligned with the guidelines proposed in this study.

In 2021, the vendor risk management and assessment organization UpGuard published a research article, unraveling the exposure of 38 million data records from multiple PowerApps portals [41]. According to Microsoft, PowerApps Portals are an extension of PowerApps, that enables citizen developers to build external facing-websites that allow users to sign in with a wide variety of identities, create and view data in Microsoft Dataverse, or even browse content anonymously [48]. The incident was covered by several IT blogs and magazines like [49] and [50], and got a lot of deserved attention, given that the leaked data included names, email addresses, Covid-19 vaccination appointments and even Social Security numbers. It is worth noting that this is a safety incident since it is not clear whether this data was exposed to any threat actors, and the portals in question could reveal this information to any user via simple browser searches.

In order to be able to model this, the root of the incident needs to be understood first, as well as which of the identified risk factors it relates to. Within PowerApps portals there is the possibility for configuration and usage of OData (Open Data Protocol) APIs, to retrieve information from the Tables stored in the Microsoft Dataverse, where the application's data is. Now, Table Permissions is the feature that imposes access limitations on the data of the Tables and it was (until prior to this incident) turned OFF by default. This means that developers had to manually turn this feature ON, and configure the desired access rules, if they wanted to avoid anonymous access to the Tables' data. Although there was a warning message shown by the platform regarding anonymous access, this was evidently not enough to avoid the incredibly serious consequences of a data breach, and guide the developer to the right direction. It is now clear that the root of the incident was a security misconfiguration, or more precisely, the absence of a necessary security configuration. Adding to this, the data seems to be stored in a readable form, judging by the example records presented by UpGuard, hinting at the absence of encryption and maximizing the damage done by the data breach.

With all the aforementioned in mind, the model is constructed, as shown in Figure 2.

The model follows the approach outlined in [11] and it showcases the activities taking place at each level alongside two severity indices. At each level, a tuple can be seen at the left hand side of the model of the form (h, s), where h denotes the human safety category and s denotes the security category. The possible values are as follows: 1 human-safety is only a bit in danger, 2 human-safety is in danger, 3 human-safety is highly in danger, and for the security levels 1 security is only slightly endangered, 2 security is endangered, 3 security is highly endangered [11]. This same ranking is used for this work because it holistically describes both concepts and perfectly fits this analysis. In this case human safety does not strictly regard physical safety but the overall effect the incident can potentially have to a person's well-being. The factors related to the incident are represented by the grey color rectangles and the human level is represented by the blue rectangle.

The root of the model is of course the security misconfiguration, and specifically the default setting of the Table Permissions feature to OFF. This has no effect on human safety, hence the score for the category is 0 and slightly endangers security since no other part of the application has been yet affected. In the Information Level (level 1) there are two conditions present. The insecure data storage resulting from the anonymous access to the data and a new factor of the information layer related to encryption. The score of the

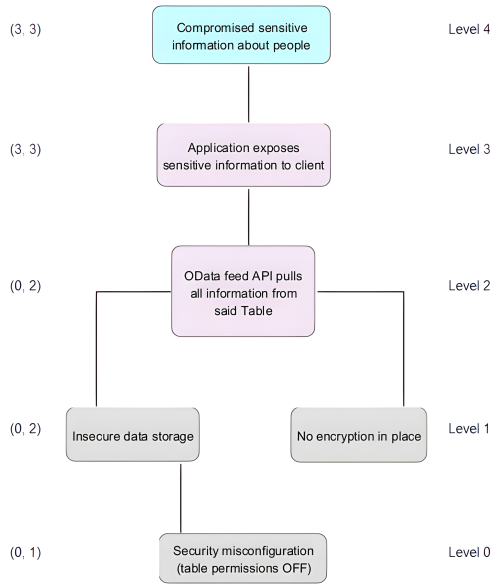

Figure 2: Multi-level Model for Microsoft PowerApps breach

security category is increased since the state of data used for the operation of the application is now affected. In the Sub-system Level (level 3), an API request is made by a client and the OData API in place pulls all the available data of the requested Table unfiltered. The scores in this level remain the same since the effect is still only internal and the data has not yet been shared externally. In the System Level, the application displays all the information to the client. This is the incident's turning point, it compromises the security and safety of the system as a whole, violating basic security principles. In this level, the human score also becomes a 3, since by the moment the information is made publicly available, the people whose data are leaked can be highly affected. Similarly, in the Human Level (level 4), the compromised information about individuals is included, once again with both human and security scores of 3 as both are fully compromised. A large-scale data breach can have numerous consequences, not only on an organizational but also on an individual level. Especially when it comes to sensitive, personal-identifiable data like home addresses, medical history data or Social Security numbers, it can lead to lack of privacy, making

the individual more vulnerable to dangers like identity theft or other fraud, as well as social matters like discrimination ([51]), or even physical safety.

This analysis perfectly showcases how this multi-level model can be a valuable tool to assess, manage and trace the risk involved in low-code development. The incident itself should serve as a warning, regarding how risky the use of ready-made application programming interfaces to interact with data can be, and how trust to the platform vendor should be limited. Concluding, it should be pointed out that it resulted in further action by Microsoft, namely in a change to the default settings of the platform for the Table Permissions feature to ON, and the introduction of a Portal Checker tool, enabling developers to detect data sets that allow anonymous access [41].

6 VALIDATION

The purpose of this section is to outline the process and outcomes of the validation phase of the study.

To ensure the validity and relevance of the results, as well as investigate the degree to which the case study findings reflect the real-world practice of low-code development, expert interviews were conducted. The results of the interviews are presented below, together with the contribution of these results to the formation of the ultimate outcome of this study; the developer guidelines.

For the above purpose two experts were interviewed, one from the field of security and one from the field of low-code development. The inclusion of at least one expert from each domain is necessary to make sure that aspects from both domains are covered in a rigorous manner. The experts have more than 10 years of experience in their respective fields, making them highly knowledgeable and credible. The interviews were brief, semi-structured interviews aiming to evaluate the study results. Semi structured interviews are helpful, when the interviewer is certain about the topics to be addressed (in this case the research findings) but still aims to allow interviewees to raise other important issues from their perspectives, add to the findings, while leaving room for clarifications [52]. The focus of the sessions was the results of the case study (Table 1, Table 2), since they capture the essence of the study, and are the foundation of all additional analysis and modeling that was conducted. The questions asked during the interviews covered evaluative aspects like the extent to which the findings reflect the main risk sources experienced in low code/the main aspects of safety or security in software development and other more field specific questions.

Interviews outcomes:

The first interview was conducted with an expert in the field of security, specifically working with certain aspects of human behaviour that are directly linked to security, aiming to identify countermeasures/interventions that make end users take safer and more conscious cybersecurity decisions. The second interview was conducted with an expert in the field of low-code development, mainly working with the Mendix and Emagiz platforms and with more than 15 years of experience in the field and also experienced with high-code development.

In the first interview the dynamics between safety and security were discussed. The value of this is an aspect that this study aims to highlight, and the interview validated that indeed the two concepts are intertwined. It was also mentioned that they are rarely considered together in practice, bringing additional value to this study's objectives. Diverse user profiles were also discussed, with regards to how they can best be supported by the LCDP, hinting to the need for further investigation into how platforms support developers, which is a future work prospect of this research.

Key outcomes of the case study were verified during both sessions. Specifically, in the second interview, the role of the Requirements Engineering phase as a significant source of risk, the relevance of common web application vulnerabilities and attacks occurring in low-code development were verified. Additionally, the first interviewee emphasized the importance of considering non-technical aspects in development, particularly concerning safety and security. These points validate the dual nature of the findings and strengthen the presented division and analysis.

The interviews also revealed certain factors that were not directly accounted for in the findings of the case study. In the first interview, the risk imposed by of an overlook of the platform's pointer was mentioned. This goes beyond the lack of vendor support factor identified, and focuses on the risk that could be inherent to the developer's actions. Another important aspect that surfaced was how the use of many different building blocks in low code development, can be dangerous for security. In the findings this is represented only in an architectural level, but the interviewee expressed this in terms of data privacy and information being handled by the application. A final addition that surfaced in the second interview, was the absence of maintenance as a risk factor. Although product aging/obsolescence is included in the findings, the root of such a risk is not included, and it is the absence of maintenance. More specifically, an application could become insecure as time passes, given that it is not adapted to the changes of the world around it, and this is regardless of how safe and secure it was at the time of development or release. For example, the introduction of new regulations, like the GDPR, require essential changes, like library updates, even in applications that already followed the best practices. Another scenario mentioned, was the possibility of certain vendors to become defunct, which calls for a careful joint process between vendor and client, in order to keep the application securely operational. This is connected to technical debt, whose diversity and unavoidable nature in low-code was also discussed. These were incorporated in the presented developer guidelines, as they are of key relevance for this research.

In both interviews, the importance of the developer and user profile was of focus, as well as how this can pose additional risk to the system. This is also reflected in the findings by the inclusion of numerous non-technical factors that are connected to the developer profile, actions and awareness level. Another interesting common outcome was the hesitant stance towards citizen development and the need for involvement of IT personnel to ensure the development of safe and secure applications. This of course depends on the scale

of the application and the information being handled in the system, as data privacy is a dominant concern in low-code; both in practice and in literature. Peer reviews for security requirements in the field of low-code (as also in high-code) was mentioned as a best practice to be followed, by both interviewees.

Overall the interviews validated the findings, but also the research objectives and scientific value of the joint analysis of the involved domains. They also revealed relevant key point for this research, enhancing the presented results.

7 DISCUSSION, REFLECTION & FUTURE WORK

This section starts with a discussion of the findings, followed by a critical reflection of the entire research project alongside potential futyure work and finally, the developer's guidelines are presented as the ultimate outcome of this work.

7.1 Discussion of Findings

This research aimed to identify and model safety risks and security threats of low-code development, providing insights to developers in the form of guidelines. The research question that has been answered is:

What are the main safety risks and security threats involved in low-code software development and how can they be modeled to best provide insights for developers?

To answer the first part of the research question a case study was conducted.

Even though in low-code development the focus is shifted from hard coding to the more business-oriented issues involved (like requirement engineering and business processes), the findings revealed that many risk factors can still occur in the requirement engineering phase. Another primary finding was that the safety and security of a developed application, highly depends on the developer's profile. This can be seen from the numerous identified factors that entirely depend on the developer's actions and choices (e.g. misconfigurations, negligence of non-functional requirements). This, in turn, can impact the safety and security of the system. As expected and also seen in the results, the platform vendors also play an important role in the safety and security of resulting systems when it comes to complexity, features and provided support. It was also shown that common web application vulnerabilities can also be present in low-code development (e.g. injection-based threats) and that additional overall risk is imposed by the off-the-shelf nature of LCDPs. Related factors include imposed black box testing, no access to source code and vendor lock-in.

LCDPs are multi-level systems, so a model that utilizes this structure was used to better showcase the findings and provide further insights. From the multi-level model it was clear that most risk factors relate to the overall system (Level 3). This is attributable to the inherent structure and design of LCDPs, wherein lower levels entail reduced control and greater reliance on preconfigured software, thereby affording developers predominant control at the highest tier. It can also be observed that most factors concern security, which is aligned with the nature of non-critical software applications.

7.2 Reflection & Future Work

All in all, the objectives of the study were met, and a concise set of results is presented. Nonetheless, reflecting on the outcomes reveals certain areas requiring improvements. Regarding the case study, the research focuses solely on the developer's perspective, making the resulting guidelines applicable for developers in practice, but requiring caution when considering other stakeholders and external factors. Broadening the scope of the search could enhance the findings, providing an overview to these aspects for all involved stakeholders (e.g. the organization, management, users etc.). These external factors and stakeholders could still have a significant impact in the safety or security of a system being developed. Furthermore, the validation process has identified the potential for external changes to impact the security or safety of developed systems, warranting further investigation. Lastly, the outcomes of the case study constitute an identified area for improvement in order to better account for risk factors that occur in low-code practice that were not cited in the literature (like the ones mentioned in the interviews, see section 6). This omission may be attributed to the researcher's lack of practical experience with low-code development.

The multi-level model enabled the clear presentation of the findings and their categorization, fitting perfectly in this work. Additionally, the observed interactions between safety and security were analyzed, however they are not visually incorporated in the model. This calls for further exploration as to how these relationships can be visually presented in such a multi-level model, as a future expansion. The used scenario fit the purpose of this research excellently, outlining the importance of the findings in practice and the capabilities of such a model.

Regarding validation of the results, the focus was on the findings of the case study. Validation and critical discussion of the developed models is not rigorously covered. This can also include the future direction, of exploring how the development and use of such models can best be integrated within the SDLC. A final remark for future exploration that was discussed in the validation process, regards the investigation into the diversity of user profiles and how they can best be guided towards safer decisions.

The study has certain scope limitations as it only included risks from specific types of software development. A broader review could reveal additional factors that are relevant to low-code development. Additionally, this work does not go beyond the developer, so it does not take into account other external aspects tied to development, like the social or organizational context, that could also contribute to the development of risky or unsafe systems through low-code practices. Further research focusing on these aspects would provide an even more complete analysis of the risk and threats in low-code software development.

With all this in mind, the guidelines for developers were formed. Integrating all key aspects of the findings and accounting for the

parts that were not directly covered, the guidelines aim to serve as a starting point for developers to critically think about the safety and security of the low-code development process and the resulting application.

7.3 Guidelines

Mapping all the identified risks and risk factors to more generic concepts combined with the gathered expert input, allows the creation/generation of certain guidelines to support the low-code development process. Overall, it is crucial to prioritize security by design throughout the entire SDLC. The resulting guidelines are mentioned in the list below. It should be noted that these guidelines may concern different stakeholders depending on the organizational context of the project (for example a company may have a group dedicated to post release monitoring), but for the sake of this research, it assumed that developers are also involved in all the mentioned tasks.

Developer's Guidelines:

- Aim for a structured and holistic requirement engineering process; it is the foundation of every project
- Focus on configuration management in all levels of the system and all phases of the Software Development Lifecycle
- Review of security requirements done by IT personnel or other knowledgeable peers for objective results and knowledge sharing
- Take platform's security pointers seriously into account when developing, but without solely relying on them to secure your system
- Avoid shortcuts for platform errors and temporary functionality
- Conduct risk assessment(s) and threat modelling prior to the Coding Phase of both the overall system/application and the used third party components
- Make sure the existing architecture that the developed system will be integrated in is well understood
- Plan time for platform selection and familiarisation with the chosen platform
- Critically evaluate vendor claims and the circumstances under which they are true
- Monitor bugs, threats and failure even post release
- Plan the implementation of version upgrades
- Aim to follow the field's best practices wherever possible
- Do not compromise rigorous testing in the face of rapid deployment
- Raise security awareness and provide security training to users
- Ensure that the developed application is adapted to the ever-changing technological ecology around it

8 CONCLUSION & CONTRIBUTION

The following part concludes the document by highlighting what was achieved, and the contribution of this work to the existing body of knowledge.

This research addressed an identified gap in the literature, regarding the risks and threats associated with low-code software development.

Research has been carried out regarding both safety and security in software development. Even though limited, some research has also been done in the domain of low-code application development. However, a joint analysis concerning both areas is still missing. Given that the usage of LCDPs is rising, the need for more research in this domain has become urgent. This research addressed this gap, by providing the following contributions; 1) identification of which safety risks and security threats involved in software development, apply to low-code practices, 2) joint modeling these risks and threats through the use of a multi-level model and a scenario, 3) categorization of the identified factors between safety and security and analysis of their interactions 4) guidelines reflecting the identified points of awareness for developers. Additionally, it contributes to the domain by serving an initial platform for designing a virtual tutoring system for citizen developers, like the one proposed in [53], specifically focusing on security aspects. The use of the multi-level approach in combination with the findings of the case study, adds value to the research and makes the followed approach unique. This constitutes an important stepping stone towards the right direction of the field; the safe and secure creation and deployment of applications, through the use of low-code practices.

9 ACKNOWLEDGMENTS

I want to thank my supervisors, Christina Kolb and Gayane Sedrakyan, for their continuous support and for sharing their expertise. I also want to thank the experts that dedicated their time to participate in the interviews, forming a crucial part of this work.

APPENDIX

"During the preparation of this work the author used no artificial intelligence tools."

Table 1: Non-technical risk factors

SDLC Phase	Risk/risk factor/threat/vulnerability	Source
Requirement Engineering	- Ambiguous/incomplete/conflicting/poorly defined/changing requirements - Unclear project objectives - Lack of COTS-driven requirements engineering process - Negligence of non-functional requirements - Neglected security requirements (and their negotiation, management, validation and prioritization etc.) - Lack of threat modeling development - Lack of security requirements awareness in customers/users	Menezes et al. (2018), Meckenstock (2024), Kusumo et al. (2012), Khan et al. (2022b)
Design	- No/inadequate/inaccurate/unrealistic planning - Complicated licensing arrangements - Security concerns in selection - Requirements mismatch with OTS selection - Too much time spent on selection - Overly optimistic learning curve - Lack of early analysis of system quality - Inadequate assumptions - Improper risk assessment/analysis/improper risk evaluation from third party components - Lack of building abuse case models and attack patterns, data flow diagram	Menezes et al. (2018), Meckenstock (2024), Kusumo et al. (2012), Khan et al. (2022b)
Coding	- Lack of difference between developers' roles and security reviewer roles to have objective results	Khan et al. (2022b)
Testing		
Deployment		
Maintenance (/post-release)	- Reduced control of future evolution of system - Product aging/obsolescence - Vendor lock-in	Kusumo et al. (2012)
Entire SDLC	- Poor/limited specifications/documentation - Faulty vendor claims - Overly optimistic expectations of quality - Vendor/organization not providing enough support/training (including security training) - Loss of cross functionality of IT developers - Distracting workflow interactions - Negligence of non-functional requirements - Lack of formal guidelines (context)/not following security design phase principles - Neglected security requirements (and their negotiation, management, validation and prioritization etc.)	Menezes et al. (2018), Kusumo et al. (2012), Meckenstock (2024), Khan et al. (2022b)

Table 2: Technical risk factors

SDLC Phase	Risk/risk factor/threat/vulnerability	Source
Requirement Engineering		
Design	- Shortfalls in externally furnished components/third party resources (unknown security, difficulty to identify whether defects are yours or the component's/add ons, using components with known vulnerabilities) - Unknown features and quality	Menezes et al. (2018), Kusumo et al. (2012), Sadqi and Maleh (2020), Silva et al. (2016)
Coding (including potential vulnerabilities of resulted app)	- Technical complexity/added complexity of unused features - Component not sufficiently adapted to changing reqs - Difficulty to upgrade to latest version - Version upgrade during development - Negative impact of component upgrade in system operability - Difficult introduction of new features/Extensibility limitations - Security misconfigurations - Interception attacks (Man In The Middle/Man in The Browser attack vulnerability) - Injection-based threats (client/server side, SQLi, XSS, command) - Business logic threats - Use of weak password-based systems - Insecure data storage - Information leakage/insecure or sensitive data exposure - Authentication & authorization (Unauthenticated key exchange, broken/missing authentication/authorization) - Encryption/Failure to use cryptographically strong random numbers - Input/output validation - Cross-site request forgery - Brute force - Logout incorrectly implemented - Timing attack (side channel attacks) - Phishing or scam - Tampering attacks	Menezes et al. (2018), Kusumo et al. (2012), Meckenstock (2024), Rokis et al. (2022), Sadqi and Maleh (2020), Silva et al. (2016), Howard, M., LeBlanc, D., & Viega, J. (2005), Chaudhari et al. (2014), Khan et al. (2022b), Nirmal et al. (2018), Gollmann (2009)
Testing	- Lack of proper tests/low test coverage/insufficient testing/lack of unit testing/Lack of testing of non-functional requirements - Imposed black box testing - Lack of final security review	Menezes et al. (2018), Meckenstock (2024), Khan et al. (2022b), Rokis et al. (2022), Kusumo et al. (2012)
Deployment	- Architectural mismatches/inadequate architecture - Lack of internal system integration/difficulty with integration with other systems	Menezes et al. (2018), Kusumo et al. (2012), Meckenstock (2024)
Maintenance (/post-release)	- Difficulty to upgrade to latest version [two phases] - Insufficient logging and monitoring of threats - Failure to handle errors - Considerations of scalability	Kusumo et al. (2012), Sadqi and Maleh (2020), Howard, M., LeBlanc, D., & Viega, J. (2005), Khan et al. (2022b), Rokis et al. (2022)
Entire SDLC	- Technical debt - No access to source code	Meckenstock (2024), Howard, M., LeBlanc, D., & Viega, J. (2005), Rokis et al. (2022)

REFERENCES

[1] Joe Mckendrick and Research Analyst. THE RISE OF THE EMPOWERED CITIZEN DEVELOPER 2017 LOW-CODE ADOPTION SURVEY. Technical report, 2017. URL www.unisphereresearch.com.

[2] What is low-code?, . URL https://www.ibm.com/topics/low-code#:~:text=Explore%20IBM's%20low%2Dcode%20solutions,applications%20through%20minimal%20hand%2Dcoding.

[3] What is Citizen Developer? | Mendix Glossary, . URL https://www.mendix.com/glossary/citizen-developer/.

[4] Stefano M. Nicoletti, Marijn Peppelman, Christina Kolb, and Mariëlle Stoelinga. Model-based joint analysis of safety and security:Survey and identification of gaps. *Computer Science Review*, 50:100597, 11 2023. ISSN 1574-0137. doi: 10.1016/J.COSREV.2023.100597.

[5] Georgios Boustras and Alan Waring. Towards a reconceptualization of safety and security, their interactions, and policy requirements in a 21st century context. *Safety Science*, 132:104942, 12 2020. ISSN 09257535. doi: 10.1016/j.ssci.2020.104942.

[6] Christoph Schmittner, Zhendong Ma, and Erwin Schoitsch. Combined safety and security development lifecylce. In *2015 IEEE 13th International Conference on Industrial Informatics (INDIN)*, pages 1408–1415. IEEE, 7 2015. ISBN 978-1-4799-6649-3. doi: 10.1109/INDIN.2015.7281940.

[7] Gartner forecast: use of low-code technologies continues to boom., . URL https://ninox.com/en/blog/gartner-forecast-use-of-low-code-technologies-continues-to-boom#:~:text=Gartner%20predicts%20that%20by%202026,not%20finding%20enough%20skilled%20workers.

[8] Miguel Lourenço, Tiago Espinha Gasiba, and Maria Pinto-Albuquerque. *You Are Doing it Wrong-On Vulnerabilities in Low Code Development Platforms*. 2023. ISBN 9781685581138.

[9] Vaishali S. Phalake and Shashank D. Joshi. Low Code Development Platform for Digital Transformation. In *Lecture Notes in Networks and Systems*, volume 190, pages 689–697. Springer Science and Business Media Deutschland GmbH, 2021. ISBN 9789811608810. doi: 10.1007/978-981-16-0882-7{_}61.

[10] Megha Quamara, Christina Kolb, and Brahim Hamid. Analyzing Origins of Safety and Security Interactions Using Feared Events Trees and Multi-level Model. pages 176–187. 2023. doi: 10.1007/978-3-031-40953-0{_}15.

[11] Christina Kolb and Lin Xie. Security and Safety in Urban Environments: Evaluating Threats and Risks of Autonomous Last-Mile Delivery Robots. 2024.

[12] Kit Siu, Heber Herencia-Zapana, Daniel Prince, and Abha Moitra. A model-based framework for analyzing the security of system architectures. *Proceedings - Annual Reliability and Maintainability Symposium*, 2020-January, 1 2020. ISSN 0149144X. doi: 10.1109/RAMS48030.2020.9153607.

[13] Panagiotis Manolios, Kit Siu, Michael Noorman, and Hongwei Liao. A Model-based framework for analyzing the safety of system architectures. *Proceedings - Annual Reliability and Maintainability Symposium*, 2019-January, 1 2019. ISSN 0149144X. doi: 10.1109/RAMS.2019.8769216.

[14] Mendix Docs, . URL https://docs.mendix.com/.

[15] Gartner , . URL https://www.gartner.com/en.

[16] Karlis Rokis and Marite Kirikova. Challenges of Low-Code/No-Code Software Development: A Literature Review. pages 3–17. 2022. doi: 10.1007/978-3-031-16947-2{_}1.

[17] Rafiq Ahmad Khan, Siffat Ullah Khan, Habib Ullah Khan, and Muhammad Ilyas. Systematic Literature Review on Security Risks and its Practices in Secure Software Development. *IEEE Access*, 10:5456–5481, 2022. ISSN 21693536. doi: 10.1109/ACCESS.2022.3140181.

[18] A. Fuller, P. Croll, and O. Garcia. Why software engineering is riskier than ever. In *Proceedings Second Asia-Pacific Conference on Quality Software*, pages 113–119. IEEE Comput. Soc. ISBN 0-7695-1287-9. doi: 10.1109/APAQS.2001.990009.

[19] Júlio Menezes, Cristine Gusmão, and Hermano Moura. Risk factors in software development projects: a systematic literature review. *Software Quality Journal*, 27 (3):1149–1174, 9 2019. ISSN 15731367. doi: 10.1007/S11219-018-9427-5/TABLES/13. URL https://link.springer.com/article/10.1007/s11219-018-9427-5.

[20] J. Whitmore, S. Turpe, S. Triller, A. Poller, and C. Carlson. Threat analysis in the software development lifecycle. *IBM Journal of Research and Development*, 58(1): 1–6, 1 2014. ISSN 0018-8646. doi: 10.1147/JRD.2013.2288060.

[21] Yassine Sadqi and Yassine Maleh. A systematic review and taxonomy of web applications threats. *Information Security Journal: A Global Perspective*, 31(1): 1–27, 1 2022. ISSN 19393547. doi: 10.1080/19393555.2020.1853855. URL https://www.tandfonline.com/doi/abs/10.1080/19393555.2020.1853855.

[22] Gopal R. Chaudhari and M. Vaidya. A Survey on Security and Vulnerabilities of Web Application. 2014.

[23] K. Nirmal, B. Janet, and R. Kumar. Web Application Vulnerabilities - The Hacker's Treasure. *Proceedings of the International Conference on Inventive Research in Computing Applications, ICIRCA 2018*, pages 58–62, 12 2018. doi: 10.1109/ICIRCA.2018.8597221.

[24] Jan Niklas Meckenstock. Shedding light on the dark side – A systematic literature review of the issues in agile software development methodology use. *Journal of Systems and Software*, 211:111966, 5 2024. ISSN 0164-1212. doi: 10.1016/J.JSS.2024.111966.

[25] Tracey Caldwell. Taking agile development beyond software – what are the security risks? *Network Security*, 2015(12):8–11, 12 2015. ISSN 1353-4858. doi: 10.1016/S1353-4858(15)30110-0.

[26] Dana S. Kusumo, Mark Staples, Liming Zhu, He Zhang, and Ross Jeffery. Risks of off-the-shelf-based software acquisition and development: A systematic mapping study and a survey. *IET Seminar Digest*, 2012(1):233–242, 2012. doi: 10.1049/IC.2012.0031.

[27] Dieter Gollmann. Software Security – The Dangers of Abstraction. pages 1–12. 2009. doi: 10.1007/978-3-642-03315-5{_}1.

[28] Amanda Bolderston. Writing an Effective Literature Review. *Journal of Medical Imaging and Radiation Sciences*, 39(2):86–92, 6 2008. ISSN 19398654. doi: 10.1016/j.jmir.2008.04.009. URL http://www.jmirs.org/article/S193986540800057X/fulltexthttp://www.jmirs.org/article/S193986540800057X/abstracthttps://www.jmirs.org/article/S1939-8654(08)00057-X/abstract.

[29] Roberta Heale and Alison Twycross. What is a case study? *Evidence-Based Nursing*, 21(1):7–8, 1 2018. ISSN 1367-6539. doi: 10.1136/EB-2017-102845. URL https://ebn.bmj.com/content/21/1/7https://ebn.bmj.com/content/21/1/7.abstract.

[30] Phillip A. Laplante. Requirements Engineering for Software and Systems. 10 2013. doi: 10.1201/B15939. URL https://www.taylorfrancis.com/books/mono/10.1201/b15939/requirements-engineering-software-systems-phillip-laplante.

[31] Minerva Journal, Minaya Vera, Cristhian Gustavo, Mendoza Vélez, Oswaldo Vicente, Arias Vera, Irina Loreley, Ministerio De Educación Daule-Ecuador, Andrés Alexander, Bravo Vera, and Henry Fabricio. Low/No-code development platforms and the future of software developers. *Minerva*, 1(Special): 21–33, 12 2022. ISSN 2697-3650. doi: 10.47460/MINERVA.V1ISPECIAL.76. URL https://minerva.autanabooks.com/index.php/Minerva/article/view/76/236https://minerva.autanabooks.com/index.php/Minerva/article/view/76.

[32] Marien R. Krouwel, Martin Op 't Land, and Henderik A. Proper. From enterprise models to low-code applications: mapping DEMO to Mendix; illustrated in the social housing domain. *Software and Systems Modeling*, 2024. ISSN 16191374. doi: 10.1007/s10270-024-01156-2.

[33] H. James Nelson, Geert Poels, Marcela Genero, and Mario Piattini. A conceptual modeling quality framework. *Software Quality Journal*, 20(1):201–228, 2012. ISSN 15731367. doi: 10.1007/S11219-011-9136-9. URL https://www.researchgate.net/publication/220636080_A_conceptual_modeling_quality_framework.

[34] GeeksforGeeks. Vendor Lock-in, .

[35] Hana A Alsaadi, Dhefaf T Radain, Maysoon M Alzahrani, Wahj F Alshammari, Dimah Alahmadi, and Bahjat Fakieh. Factors that affect the utilization of low-code development platforms: survey study. *Romanian Journal of Information Technology and Automatic Control*, 31(3):123–140, 2021. doi: 10.33436/v31i3y202110. URL https://doi.org/10.33436/v31i3y202110.

[36] Low Code Platform Market. URL https://straitsresearch.com/report/low-code-development-platform-market.

[37] Mendix Docs. Best Practices for App Security . URL https://docs.mendix.com/howto/security/best-practices-security/.

[38] Michael Bargury. Hackers Abuse Low-Code Platforms And Turn Them Against Their Owners, 10 2021.

[39] GeeksforGeeks. Black Box Testing, . URL https://www.geeksforgeeks.org/software-engineering-black-box-testing/.

[40] Karlis Rokis and Marite Kirikova. Exploring Low-Code Development: A Comprehensive Literature Review | Rokis | Complex Systems Informatics and Modeling Quarterly, 2023. URL https://csimq-journals.rtu.lv/article/view/csimq.2023-36.04.

[41] UpGuard Team. By Design: How Default Permissions on Microsoft Power Apps Exposed Millions, 2021. URL https://www.upguard.com/breaches/power-apps.

[42] Gartner. Technical Debt, . URL https://www.gartner.com/en/information-technology/glossary/technical-debt.

[43] Gartner. Enterprise Low-Code Application Platforms Reviews and Ratings, . URL https://www.gartner.com/reviews/market/enterprise-low-code-application-platform.

[44] Mendix. Native Back Button. URL https://marketplace.mendix.com/link/component/114055.

[45] Mendix Security, . URL https://docs.mendix.com/refguide/security/.

[46] Mendix Logout, . URL https://docs.mendix.com/appstore/partner-solutions/ats/rg-one-logout/.

[47] Jaspreet Bedi and Kuljit Kaur. Understanding factors affecting technical debt. *International Journal of Information Technology (Singapore)*, 14(2):1051–1060, 3 2022. ISSN 25112112. doi: 10.1007/S41870-020-00487-9/TABLES/6. URL https://link.springer.com/article/10.1007/s41870-020-00487-9.

[48] Someleze Diko. Differences between PowerApps Portals and Power Pages, 2022. URL https://techcommunity.microsoft.com/t5/educator-developer-blog/differences-between-powerapps-portals-and-power-pages/ba-p/3571229#:~:text=PowerApps%20Portals%20are%20an%20extension,or%20even%20browse%20content%20anonymously.

[49] Scott Ikeda. Power Apps Data Leak, 8 2021. URL https://www.cpomagazine.com/cyber-security/microsoft-power-apps-data-leak-fallout-38-million-records-exposed-state-and-city-governments-among-those-breached/.

[50] Lily Hay Newman. 38M Records Were Exposed Online—Including Contact-Tracing Info. 2021. URL https://www.wired.com/story/microsoft-power-apps-

data-exposed/.

[51] Ningning Wu and Robinson Tamilselvan. A Personal Privacy Risk Assessment Framework Based on Disclosed PII. *Proceedings - 2023 7th International Conference on Cryptography, Security and Privacy, CSP 2023*, pages 86–91, 2023. doi: 10.1109/CSP58884.2023.00021.

[52] Chauncey Wilson. Semi-Structured Interviews. In *Interview Techniques for UX Practitioners*, pages 23–41. Elsevier, 2014. doi: 10.1016/B978-0-12-410393-1.00002-8.

[53] Gayane Sedrakyan and Monique Snoeck. Lightweight semantic prototyper for conceptual modeling. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8823: 298–302, 2014. ISSN 16113349. doi: 10.1007/978-3-319-12256-4{_}32.

[54] Davide Di Ruscio, Dimitris Kolovos, Juan de Lara, Alfonso Pierantonio, Massimo Tisi, and Manuel Wimmer. Low-code development and model-driven engineering: Two sides of the same coin? *Software and Systems Modeling*, 21(2):437–446, 4 2022. ISSN 1619-1366. doi: 10.1007/s10270-021-00970-2.

[55] Yajing Luo, Peng Liang, Chong Wang, Mojtaba Shahin, and Jing Zhan. Characteristics and challenges of low-code development: The practitioners perspective. In *International Symposium on Empirical Software Engineering and Measurement*. IEEE Computer Society, 10 2021. ISBN 9781450386654. doi: 10.1145/3475716.3475782.

[56] Lissette Almonte, Universidad Autónoma de Madrid Madrid, Spain Iván Cantador, Spain Esther Guerra, Spain Juan de Lara, Iván Cantador, Esther Guerra, and Juan de Lara. Towards automating the construction of recommender systems for low-code development platforms. *Proceedings - 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS-C 2020 - Companion Proceedings*, pages 451–460, 10 2020. doi: 10.1145/3417990.3420200. URL https://dl.acm.org/doi/10.1145/3417990.3420200.

[57] Megha Quamara, Christina Kolb, and Ankur Lohachab. Where do Safety and Security Mutually Reinforce? A Multi-level Model-based Approach for a Consistent Interplay. 2024.