UNIVERSITY OF LJUBLJANA
SCHOOL OF ECONOMICS AND BUSINESS

MASTER THESIS

LOW-CODE SOFTWARE DEVELOPMENT PLATFORMS IN INSURANCE INDUSTRY

Ljubljana, July 2023 EVA GASHTEVSKA

AUTHORSHIP STATEMENT

The undersigned Eva Gashtevska, a student at the University of Ljubljana, School of Economics and Business, (hereafter: SEB LU), author of this written final work of studies with the title Low-code software development platforms in insurance industry, prepared under supervision of red. prof. dr. Tomaž Turk

D E C L A R E

1. this written final work of studies to be based on the results of my own research;
2. the printed form of this written final work of studies to be identical to its electronic form;
3. the text of this written final work of studies to be language-edited and technically in adherence with the SEB LU's Technical Guidelines for Written Works, which means that I cited and / or quoted works and opinions of other authors in this written final work of studies in accordance with the SEB LU's Technical Guidelines for Written Works;
4. to be aware of the fact that plagiarism (in written or graphical form) is a criminal offence and can be prosecuted in accordance with the Criminal Code of the Republic of Slovenia;
5. to be aware of the consequences a proven plagiarism charge based on the this written final work could have for my status at the SEB LU in accordance with the relevant SEB LU Rules;
6. to have obtained all the necessary permits to use the data and works of other authors which are (in written or graphical form) referred to in this written final work of studies and to have clearly marked them;
7. to have acted in accordance with ethical principles during the preparation of this written final work of studies and to have, where necessary, obtained permission of the Ethics Committee;
8. my consent to use the electronic form of this written final work of studies for the detection of content similarity with other written works, using similarity detection software that is connected with the SEB LU Study Information System;
9. to transfer to the University of Ljubljana free of charge, non-exclusively, geographically and time-wise unlimited the right of saving this written final work of studies in the electronic form, the right of its reproduction, as well as the right of making this written final work of studies available to the public on the World Wide Web via the Repository of the University of Ljubljana;
10. my consent to publication of my personal data that are included in this written final work of studies and in this declaration, when this written final work of studies is published.
11. that I have verified the authenticity of the information derived from the records using artificial intelligence tools.

Ljubljana, ________________________ Author's signature: _________________________

(Month in words / Day / Year, e. g. June 1st, 2012

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF APPENDICES

LIST OF ABBREVIATIONS

sl. – Slovene

LCDP - (sl. malokodno razvojno okolje); Low-code Development Platform

NCDP - (sl. brezkodno razvojno okolje); No-code Development Platform

UI – (sl. uporabniški vmesnik); User Interface

GUI – (sl. grafični uporabniški vmesnik); Graphical User Interface

OMG - Object Management Group

MDD – (sl. modelno vodeni razvoj); Model-Driven Development

MDA – (sl. modelno vodena arhitektura); Model Driven Architecture

CIM – (sl. računsko neodvisni model); Computation Independent Model

PIM – (sl. model neodvisen od računalniškega okolja); Platform Independent Model

PSM – (sl. model odvisen od računalniškega okolja); Platform Specific Model

BPMN – (sl. notacija modeliranja poslovnih procesov); Business Process Modeling Notation

DMN – (sl. notacija odločitvenih modelov); Decision Model and Notation

IDE – (sl. integrirano razvojno okolje); Integrated Development Environment

API – (sl. aplikacijski programski vmesnik); Application Programming Interface

PaaS – (sl. platforma kot storitev); Platform as a service

RAD – (sl. hitri razvoj programskih rešitev); Rapid Application Development

AI – (sl. umetna inteligenca); Artificial Intelligence

ML – (sl. strojno učenje); Machine Learning

INTRODUCTION

In today’s world, technological development is progressing at a high pace, making people’s lifes simpler. Innovations on the market appear more frequently than before, prompting companies to follow the newest trends in order to be ahead of their competitors. One of these innovations, which is in its early phase of popularity, is the low-code development platform (LCDP).

LCDPs are platforms that enable developers and people with little experience in development to develop applications with minimal writing of code. Their main advantage is the efficiency of creating a working application, which enables higher productivity, lower costs, easier maintenance of applications and involvement of the stakeholders in the development process (Talesra & Nagaraja, 2021). Considering these benefits, LCDPs became a very attractive solution for many insurance companies, especially since digitalisation caused many insurance companies' systems to become outdated and hard to maintain. By using LCDPs, the dependency of insurance companies on highly skilled developers is lowered. This means that businesspeople in insurance companies could create new insurance products or change the existing ones with or without the developer’s help, making big room for innovations that can be created in a short period of time (Daly, 2020).

On the market for LCDPs, there are many companies racing to offer the best LCDP. Despite similar architecture and processes of the platforms, the suppliers’ market focus could be different – some are enterprise platforms, and others are based on a specific industry. Enterprise platforms are general platforms because they can be used for building applications for any industry. In the master’s thesis, I will identify the advantages and disadvantages of two LCDPs. The first platform is Mendix, which is considered a general platform for building any kind of application. There are three main reasons why I chose Mendix for analysis in my master thesis. Firstly, this platform is positioned among the leaders with the highest completeness of vision in the magic quadrant for enterprise LCDPs from Gartner for 2020 (Gartner, 2020). Secondly, Mendix has published many use cases of insurance companies that used Mendix for the development of products. Thirdly, Mendix was used by Adacta, the company that developed the AdInsure platform – the second LCDP chosen for the analysis. Mendix was used for the development of a mobile application for insurance quote integrated with the AdInsure platform, which confirms Mendix’s compatibility with the insurance industry.

AdInsure, which is the second LCDP, has a Graphical User Interface (GUI) based low-code tool called AdInsure Studio. It enables business users and IT professionals in insurance companies to implement changes in insurance processes and products. I chose AdInsure because of my experience working with the AdInsure Studio and the easy access to sources needed for the analysis. Additionally, the LCDPs specialised for particular industries are not free of charge, which makes them difficult to access.

The purpose of the master's thesis is to contribute to the understanding of LCDPs and their usage in the insurance sector. The main goal of the master's thesis is to analyse the functionalities of industry-based and general LCDPs, compare them and find the main differences by developing a business software solution called Home insurance product in the selected platforms.

Home insurance is a type of insurance that covers loss and damage caused to a property by a harmful event. The financial protection is related to the insured object, which is the building and could also cover attached buildings, such as a garage, personal belongings within the buildings, costs associated with a person's damage that occurred on the insured property, such as injury, and additional living expenses that appeared because of the loss or damage of the insured property. Home insurance can be purchased by individuals who own a home, rent properties, and landlords (Understand Insurance, n.d.). The person that purchases Home insurance is called a policyholder. Terms related to the insurance package are specified in an insurance policy that represents a legally binding contract between the insurance company and the policyholder. The cost of obtaining Home insurance is called premium, and the policyholder is obliged to pay a certain amount of money to the insurance company (Fabozzi & Drake, 2010). Many factors affect the insurance premium amount, which are related to the probability of occurrence of a harmful event. If the probability is higher, the premium amount will also be higher (Dorfman, 1998). I decided to develop a Home insurance product because of the availability of data about all calculations related to the insurance premium.

The main base for comparison will be the implementation of the defined business requirements into business software solution in both platforms. The business requirements for the insurance product development are gained by business analysis of the actual needs of an insurance company that offers Home insurance. They contain general information about the product as well as coverage calculation specifics, product workflow, contract participants and User Interface (UI) information, which are common inputs for all insurance products. The results from the analysis will be used to define what functionalities the insurance company needs in order to implement and maintain the mentioned insurance product by the low-code principles.

Goals:

- Define the LCDPs, their general architecture and their development process.
- Define software solution's development specifications.
- Identify the key functionalities of the LCDPs required for developing an insurance product.

Research Question: What are the key differences between using industry-oriented and general LCDPs for developing insurance solution, based on the comparison between Mendix and AdInsure platforms?

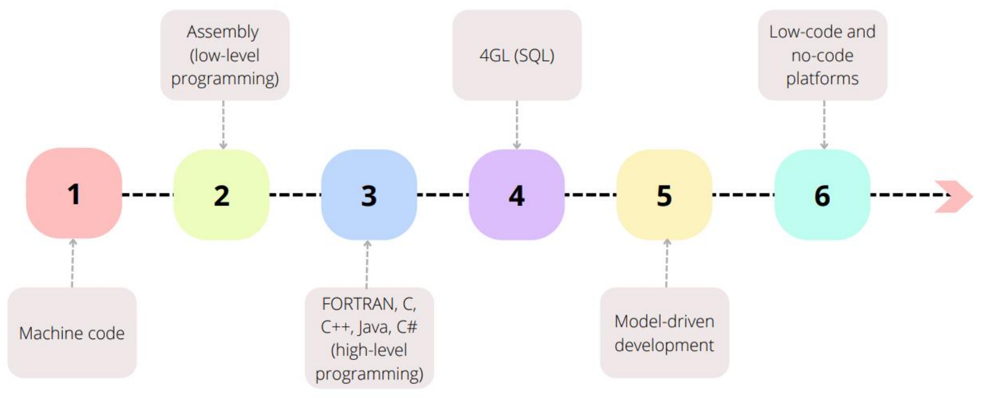
The master’s thesis contains a theoretical and a practical part. In the first three chapters, I present the theoretical part, where I use secondary sources to define the LCDPs, the main insurance terms that will be used in the practical part, trends in the insurance industry, as well a description of the analysed LCDPs. In the practical part, I will use primary sources to make a comparative analysis of the chosen LCDPs by the implementation of the defined business requirements for the Home insurance software solution. The implementation of the product will be done in each platform, and its advantages and disadvantages will be analysed in each cycle of the development process. In the end, I will identify the needed functionalities of a LCDP in order to have a complete development cycle of the chosen insurance product.

1 LOW-CODE PLATFORM

1.1 Abstraction concept in software development

Nowadays, the “low-code platform” term is classified as a new trend many companies have not considered a business opportunity due to a lack of conceptual understanding. However, this concept is not entirely an innovation that has appeared on the market in the last decade. Besides the specific practices of the LCDPs, it has a common goal, with other software development technologies introduced throughout history, to increase the level of abstraction in software development. According to that, the low-code concept can be considered as an improvement of the previous attempts to create a solution that will minimise the amount of code during the development of software applications (Bock & Frank, 2021a).

Figure 1: Abstraction stages of the development process


Assembly (low-level programming)
4GL (SQL)
Low-code and no-code platforms
1
2
3
4
5
6
Machine code
FORTRAN, C, C++, Java, C# (high-level programming)
Model-driven development

Adapted from ERP-One (2020).

The abstraction concept in the software engineering field has great importance. It represents the details of the software systems that are exposed to the programmers or other users involved in the process of software development. This means the higher level of abstraction, the lower complexity of the software systems. Throughout history, the abstraction level in programming languages has significantly increased, which led to a less error-prone process of developing new systems, decreased time spent on it, and decreased amount of written code. Moreover, by hiding many repetitive operations, the languages with higher abstraction levels widened their circle of users by including non-programmers. The progress toward the abstraction of processes is depicted in Figure 1, which is divided into six stages. The first four stages cover the evolution of programming languages, and the last two stages cover the software development methodologies of model-driven development (MDD), LCDP and no-code development platform (NCDP) (Damaševičius, 2006).

Stage 1: At the very beginning, the software development was executed in machine code (binary code in 1s and 0s), which is also known as processor language. The machine language is considered the first generation of programming languages. Apart from some pros of using machine language, such as fast processing of the code and lack of necessity for a translator, there were many cons that led to the development of the Assemble language. The most important con was the difficulty of the language, which required a lot of energy and time for people to learn the language. Additionally, a small error costed a significant amount of time for the programmer since debugging was a difficult task, and it required an understanding of the architecture of systems (Kahanwal, 2013).

Stage 2: The struggle of writing long code that is not understandable to people led to the development of Assembly language, which is a low-level programming language that is translated into machine code. Assembly is the second generation of languages and a step closer to human understanding since it offers alphanumeric symbols in English language. However, the Assembly language did not remove all obstacles found by using the machine code. Despite the improvements in the debugging field and retained efficiency in processing the code, programmers still needed a great knowledge of computer architecture. Moreover, the instructions in the Assembly language were different for every computer, which means the same set of instructions could not be used for many computers (Kahanwal, 2013)

Stage 3: However, the abstraction level was still not satisfying, which caused the emergence of high-level languages or so-called the third generation of programming languages. The first member of this group was the FORTRAN programming language, which appeared in 1956. Over time, many other programming languages, which are nowadays among the most used, joined the group, such as C, C++, Java, Python, C# and so on. The main advantage was the usage of English language in coding, so the code could be readable and understandable, which led to higher productivity of programmers (Chen, Dios, Mili, Wu, & Wang, 2005).

Stage 4: The fourth generation of programming languages, or database languages, are even more understandable to humans. Apart from being more friendly to programmers and understandable to non-programmers, they offer higher productivity in data management, reporting, graphics, and end-to-end interface. A member of this group that is widely known is SQL (Baer, 2010).

Stage 5: The progress towards abstraction did not stop here. The aim of achieving abstraction took the game to another level by introducing MDD facilitated by using models as a visual representation of the software structure. In this methodology, the models are a tool that can visualise the software solution that needs to be developed with all necessary details that are relevant to developers and business people. Additionally, they can be used to represent the current situation of an existing system and how it needs to be improved. This means that these tools can be used before writing code or after the system exists (Brown, Conallen, & Tropeano, 2005)

Stage 6: There were still negative sides to this paradigm since the developers were required to define very detailed models. This led to additional efforts and novelty on the market – LCDPs and NCDPs (Kahanwal, 2013).

The last two stages are not part of programming language evolution as described. However, these two software development methodologies had a significant impact on the software development field. In addition to the division of the MDD, and LCDPs and NCDPs as different paradigms, there are many similarities among them that have opened the question in academia if these paradigms should be considered separately. The key features of each of them are described in the next chapters as well as the common features which are the main reason for the confusion of the terms.

1.2 Model-driven development

The expectations of the customers over the years regarding software solutions have reached another level. Therefore, the need for more complex products appeared to be a new challenge for software companies. Along with the expectations for more complex functionalities, the quality of these software solutions and time for delivery were expected not to be compromised. These expectations were tackled by the MDD approach and many available tools facilitating its application (Selic, 2003).

In most cases, developing software starts with a sketch of components that need to be developed and the relationships between them. These visualisations or so-called “models” could be made from a technical point of view, serving as an artefact that facilitates communication and collaboration between developers included in the project or sketch developed by the business team to communicate the business scenario to developers. However, this practice does not bring the same level of benefits through the software development cycle because of two main reasons. Firstly, in the case of developing complex programs, the documented model at the start of the development process is never the same as the final result. In order to reach a state where the model depicts the current progress, it requires constant effort in tracking and changing the model. Secondly, the models that are of a purely technical nature limit the collaboration and communication with the other stakeholder, such as business people. This also includes technical models that are produced as a translation of the business model developed by the business people. If the model is understood by the business people, the need for transformations can be found earlier (Selic, 2003).

MDD is a methodology for software development that is based on models. These models represent how the software should be structured by incorporating the domain perspective. Based on the developed model, the source code is generated. There is a conceptual framework called MDA (Model Driven Architecture) that describes how these models must be defined in order to have traceability between the model elements' transformation during the whole cycle of software development and a possibility for automated transformation of models (Brown, Conallen, & Tropeano, 2005). This concept is developed by the Object Management Group (OMG) as one of their modelling standards for visual software design, maintenance, and implementation. It is also important to note that the MDD and MDA do not have the same meaning. The MDD is a broader concept than the MDA and includes MDA. This means that MDD can be implemented by using methodologies other than MDA (OMG, 2014)

The models are platform independent, which stresses the main characteristic and benefit of using this MDA standard – isolation of rapid changes in the technology. The developed model can be translated into platform specific model by any MDA tool. Before further explaining what MDA represents, some basic concepts need to be laid out (OMG, 2014).

- System – it generally means reaching out to areas other than the software itself. Some elements that create a set and are related to each other with the purpose of fulfilling a specific goal are creating a system (OMG, 2014).
- Model – represents a part of the system or the whole system with all terms used with a detailed description of their meaning and all rules that have to be applied. Depending on the goal, it can represent part of the system from different aspects, for example, the model of business processes or the hardware structure. Despite the system's structure, the model also specifies the system's function (OMG, 2014).
- Modelling language – a language used to describe the model that has formalised and defined meaning. This type of modelling language is called formal language. On the other hand, there is also informal language, which is used for expressing the model. However, this type of language does not have terms that are standardised, and the developer has the freedom to develop the model. This leads to issues which could cause additional costs because there is a possibility that the model will be misinterpreted (OMG, 2014).
- Platform – facilitates the implementation of the system. In the MDA glossary, the platform as a concept is not only viewed from the technical point of view representing only the software or hardware where the application will be realised, such as Microsoft

.NET. This concept is also interpreted from a business and domain perspective, and an example of a platform from that perspective are company employees (OMG, 2014).

There are three models defined by the MDA standard: Computation Independent Model (CIM), Platform Independent Model (PIM) and Platform Specific Model (PSM). Each model defines a different abstraction level (Kardoš & Drozdova, 2010).

The CIM represents the business side of a system, such as the business processes and isolates the technical terms related to the technology area. This means that the modelers are the business people, such as the business analyst or other actors with knowledge of the specific domain. Since the system in this model is presented from a pure business perspective with the main business terms, processes and relations, this model is a base for people with insufficient business knowledge. Moreover, the CIM is a starting point and prerequisite for the development of both other MDA models: PIM and PSM (Kardoš & Drozdova, 2010).

PIMs are independent of the platform that will be used for developing the system. This means that on this level of abstraction, a model is designed that depicts the set of services or, in other words, the functionality of the system without covering the technical details of the platform. For this model is used another popular standard developed by the OMG called Unified Modeling Language (UML). It is a formal modelling language that visualises the system that needs to be developed, such as activities, components, external users, and interactions between all these elements (Kardoš & Drozdova, 2010).

PSMs are the models that are specific to a platform. They are basically a transformation of the defined PIM for a specific platform. The modellers on this level are software developers, as the nature of these models is only technical. PSM includes rules and other specific technical information on how the PIM will be implemented on a specific platform. The PSM that has the last position in the chain of models is transformed into code for the platform defined in the PSM (Kardoš & Drozdova, 2010).

The MDA methodology aims for reusability that can be traced among the models, as a PSM of one system can be used as a PIM of another (Kardoš & Drozdova, 2010).

However, the concept of MDD is not the perfect choice for every software development project. Like many other methodologies, this one also has some advantages and disadvantages.

Advantages:

- Among the biggest advantages of using the MDA methodology is the higher productivity achieved by the minimised time needed for developing software (Mousami, 2014).
- The models are automatically translated into code, which means that they are less error-prone compared to manually writing code for a specific system. This characteristic is directly related to the quality of the system (Mousami, 2014).

- Since the code is generated automatically, the changes needed to be applied to the system do not require many resources, which means that the maintenance of the program is not difficult (Selic, 2003).
- As mentioned before, the MDA includes a model that is platform-independent, which is later transformed into a model for a specific platform. This means that the model can be implemented on many technological platforms and is not dependent on one, which minimises the risk of changes to a specific technology (Selic, 2003).

Disadvantages:

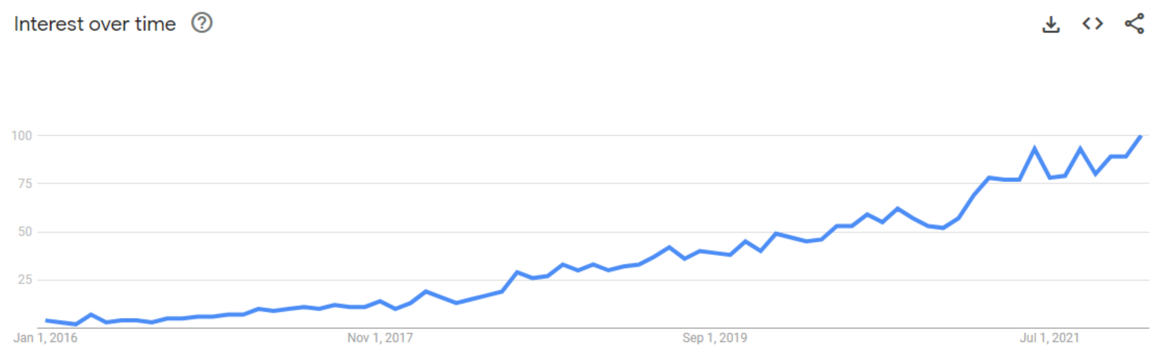
- Using MDD only for some part of the whole system cannot have a significant positive impact on productivity, as MDD methodology promises. This scenario is not so likely when there are interrelated models and one of the models is changed. Even though some changes in the model can be automatically applied to all related models, there are examples where this is not possible (Hailpern & Tarr, 2006).
- Another negative point is the complexity of the models and the impact of changes in case of interrelated models. The specialists in these cases are required to have knowledge and understand the models that they are working on and all related models that might be affected (Hailpern & Tarr, 2006).
- Companies already have their own system in which they invested a significant number of resources, so the option to switch to MDD in later phases of the development is not the most convenient one (Mousami, 2014).

In the research paper "Low-code development and model-driven engineering: Two sides of the same coin?" (Di Ruscio et al., 2022), the authors have compared MDD and low-code development and pointed out the differences and similarities. For platforms and tools falling under the same paradigm umbrella, such as MDD or low-code, it does not mean that they all have the same functionalities. There are many categories of these platforms that bring specific benefits to their users. For example, one of the known characterises of LCDPs is that they are cloud-based, but this is not the case with all LCDPs. A well-known benefit of tools that incorporate MDD is the minimisation of the amount of code. Some tools do not offer this feature since they are designed for resolving some other problem, for example, software optimisation. Additionally, the authors specified three main areas where MDD and LCDP differentiate: platform, users, and domain. The first area is concerned with whether the platform is cloud-based or desktop based. The cloud-based platforms are more present in the low-code field, and desktop-based platforms are more characteristic of MDD. The second area is about the target users of these two approaches. The focus in LCDP is on developers and citizen developer, who is a user with a business background. On the other hand, MDD focuses more on developers and users with software engineering backgrounds. The third area is the different domains that these approaches have as a target. The area that LCDP highlights is the business domain since its target is the business user. As mentioned before, the MDD targets more technical users, so consequentially, the domain is more technical-oriented, such as the automotive industry.

1.3 Low-code platforms

In 2014, Forrester, a research company in the field of technology, came up with a definition of LCDP, which specifies the benefits and the domain of the newest trend. Minimal time spent on application development, decreased amount of code, and reduced costs related to development are the main benefits of using LCDP. Moreover, the domain of these applications is business, which means the product of LCDP are business applications. With time, the number of vendors offering this kind of platform has increased, as well as the popularity of the LCDPs (Clay & Rymerwith, 2016). This is confirmed by Google Trends as displayed in Figure 2, where it is visible that the term "low-code development platform" in the Software category started its trend in late 2016 and continued gaining traction over time (Google Trends).

Figure 2: Search trend for "low-code development platform" over time



Source: Google Trends (n.d.).

However, the definition of LCDP is not strict enough; therefore, the market offers many solutions that are considered low-code (Maier, Ulrich, & Bock, 2021). This category includes Integrated Development Environment (IDE) and MDD tools (Bock & Frank, 2021b). The broad definition of LCDP makes room for companies to regard their solutions as LCDP because they are based on the low-code approach. Besides the fact that they have common features, there could be many areas where they differ from each other (Maier, Ulrich, & Bock, 2021).

LCDPs are platforms that enable developers and people with little development experience to develop applications with minimal time spent on writing code. Their main advantage is the efficiency of creating a working application that enables many benefits for the business, such as higher productivity, lower costs, easier maintenance of applications, and involvement of stakeholders in the development process (Talesra & Nagaraja, 2021). These platforms are mainly provided as cloud-based platform as a service (PaaS), which shortens the deployment cycle of the application and consequently shortens the delivery time needed for production (Sahay, Indamutsa, Di Ruscio, & Pierantonio, 2020). Additionally, the development cycle is shorter since people with little or no technical knowledge are taking on some of the developer's tasks. If we take the SCRUM methodology as an example, business users in a development company will eliminate the time spent on writing scenarios for improvements or bugs as well as eliminate the time spent on waiting for developers' availability to tackle the issue. Instead, they could make the needed improvement after some clicks (Carroll, Móráin, Garrett, & Jamnadass, 2021).

The low-code method can be spotted in many platforms and tools that were used before the appearance of the LCDPs on the market. It is related to the rapid application development (RAD) method, whose targeted user group includes business users (Pratt, 2021). RAD is characterised by the development of business applications by using a combination of computer-aided software engineering (CASE) tools, GUI builders, data management systems and programming languages from the 4th generation. Moreover, it has in common objectives as low-code development: rapid development, low costs, and development of high-quality system (Beynon-Davies, Mackay, Carne, & Tudhope, 1999). Some of the well-known RAD tools for developing applications are Excel and Microsoft Access (Pratt, 2021).

In the literature, there are various perspectives on the LCDPs. Some of the authors, such as Bock & Frank (2021a), claim that the LCDPs are not innovation in the software development field. The usage of low-code principles has been traced back to the beginning of programming languages, where the main goal was to raise the level of abstraction and more efficiently produce code. The main benefit of using LCDPs which is higher productivity, mainly comes from the integration of various tools and systems into one. However, these systems are far from innovation since their presence can be traced before the LCDPs' appearance on the market, such as in data management system or GUI tools for visual development.

Bock & Frank (2021b), in their research for low-code platforms, found out that the majority of platforms included in their study were products that existed before and were marketed under different labels such as RAD, PaaS, MDD platforms or business process management (BPM). According to them, the main capabilities of LCDPs that are most common are:

- Data modelling – the user can define the data structure visually by using a diagram, for example, in Entity-Relationship Model (ERM).
- External data sources support – LCDPs support Application Programming Interface (API) integration with other applications and systems for data access. Moreover, they also provide an internal database and the option of using an external one.
- GUI designer - it is an important feature that all LCDPs have. This allows users to choose between a variety of widgets and components in order to customise the interface of the application by drag-and-drop. These designers are not structured only for one environment, such as a desktop but offer the possibility to design the UI for other environments.

– Simple deployment – the deployment process is simple and usually done with a single click.
– Extendibility of application and integration with other applications – extending the application developed in LCDP by using services from external sources. This functionality is enabled via APIs.
– Security support – it relates to the definition and assigning specific users and restriction of access to particular pages and processes.

In addition to the common functionalities of LCDPs, there are platforms with different sets of additional functionalities that are considered their key functionalities and, at the same time, present their strengths. These platforms belong to different categories based on their functionalities. The authors Maier, Ulrich, & Bock (2021), in their research report, analysed 30 platforms that were marketed as low-code and based on that, they defined four categories. The first category is basic data management platforms, consisting of LCDPs that focus on data management provided with the help of GUI tools. The second category includes LCDPs that focus on workflow management, which means visually defining the logic of the application without writing code. The third category is the extended and GUI-centred IDEs. It contains LCDPs that are offering IDEs that provide much more development support than regular IDEs. The fourth category includes multi-use platforms for business application configuration, integration, and development. These are platforms for mainly business applications, as the category name suggests, offering a different range of features with a main focus on application lifecycle support.

Along with the low-code trend, new terms appeared that are closely related to it. One of them is the "citizen developer", characterised by people that are directly involved in the development of an application by using tools that enable application development without the need for technical knowledge. This is enabled by environment visualisation that helps business users understand the process flows and application logic, make changes and innovate without or with little high-skilled developer support. Citizen developers are the main trigger for digital transformations in companies. Moreover, their involvement in the product development process also gives them more power to control the quality of the product. However, aside from the main motive of these tools being easy to use, they cannot erase the prerequisite for basic knowledge and understanding. If companies invest in adopting intuitive tools that can be easily used by business people, they must invest in training to use these tools as well. Investment in citizen developers is reducing the risk of expanding the market gap for IT people. However, it does not exclude the need for IT professionals (Carroll, Móráin, Garrett, & Jamnadass, 2021).

An additional concept that is often used in pairs with LCDPs is the NCDP. The reason for this is the missing conceptualisation of the terms, which makes it difficult to separate these platforms in the research analysis. NCDPs are marketed as platforms that rely on visual application development, which are used for developing simple solutions and do not require the writing of any code (Di Ruscio et al., 2022). The key differences with the LCDPs are:

- Target users - LCDPs target group also includes developers; however, NCDP includes only business users (Pratt, 2021).
- Level of coding – the key feature of the NCDPs is contained in its name, which is no code. This is also the key difference between the LCDPs and NCDPs (Pratt, 2021).
- Usage – with no coding, the possibility of creating a complex application supporting some core processes is limited. The main reason is this kind of platform's limited scope of prebuild templates and connectors. On the other hand, the LCDPs can remove these limitations by expanding the platform scope with manual code (Cabot, 2020).

In practice, there is a gap between the marketed content from the companies offering this kind of product and researchers that did use NCDP for practical scenarios. The results show that developing some scenarios still requires people with technical skills or non-technical people to invest time in gaining technical knowledge through training. Another reason that supports the stance of existing one market for LCDPs and NCDPs is that NCDPs are contained in the LCDPs, which means simple applications without coding and only by visual development can also be created by using LCDPs (Pratt, 2021).

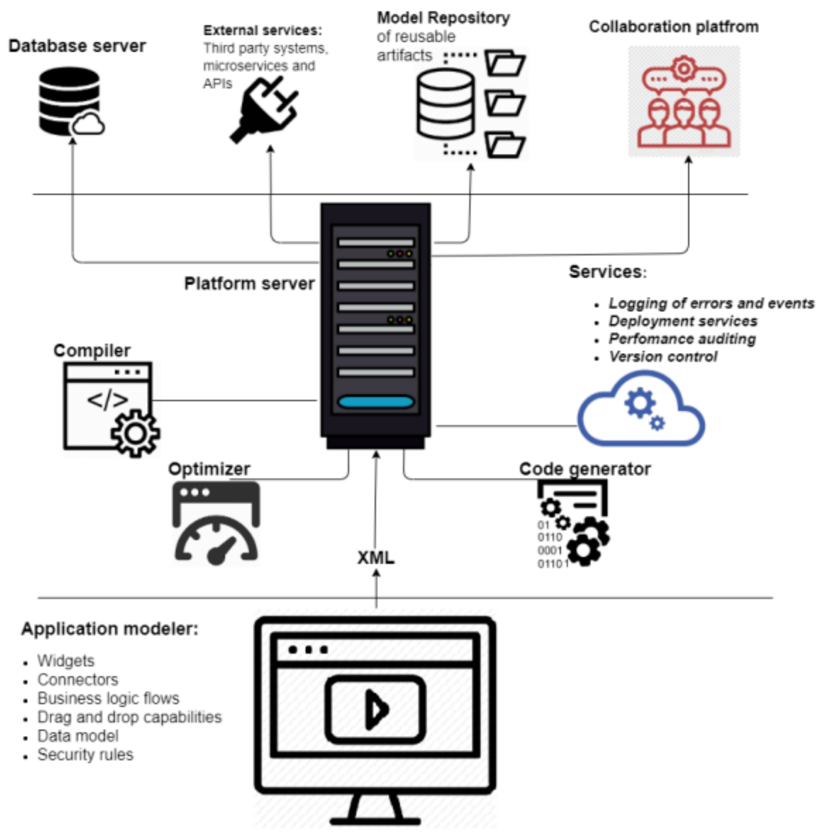
1.4 Architecture and development process in low-code platforms

As mentioned before, the LCDPs can have a scope of critical functionalities offered to the user, which can be split into different categories. Despite the various LCDPs offered on the LCDP market and their different key functionalities, LCDPs are made of the same architecture blocks.

Figure 3 shows the main components of the LCDPs. In the buttom section, we see the Application modeller, which consists of many elements that help users build the application. These elements include widgets, connectors, business logic flows, drag-and-drop capabilities, data models, and security rules. In other words, this section represents the GUI, where the user designs the UI and the application logic with the help of the mentioned elements (Sahay, Indamutsa, Di Ruscio, & Pierantonio, 2020). For example, the user designs a page with the usage of offered platform widgets by dragging and dropping, and then by using the workflow element, determines what the application will calculate if a button is clicked.

The Platform server is placed in the middle section with its main elements: the compiler, optimizer, code generator and services. Here is where the actions that the user takes from the GUI are sent and further processed (Sahay, Indamutsa, Di Ruscio, & Pierantonio, 2020). The compiler takes care of translating the actions that the user will take into a code (TechTarget, 2022), for example, the action mentioned on the button in the previous paragraph. Some LCDPs even offer access to the generated code (Sahay, Indamutsa, Di Ruscio, & Pierantonio, 2020). The optimiser, on the other hand, takes care of finding the most efficient way to run the program (Bentley, 1982). Furthermore, in this section are present many services such as logging errors and events, deployment services, performance auditing and version control (Sahay, Indamutsa, Di Ruscio, & Pierantonio, 2020).

Figure 3: Main components of low-code development platforms



Source: Sahay, Indamutsa, Di Ruscio, & Pierantonio (2020).

In the upper section are positioned all other services that the created model by the user interacts with. The first element is the internal or external database that the model uses. The second element are the micro-services and integration with external systems through APIs. The model repository is the third element which contains reusable artefacts that can be used in the new model for different purposes, such as UI, logic, or data. Finally, there is the collaboration platform that supports collaboration between different developers working on the same model (Sahay, Indamutsa, Di Ruscio, & Pierantonio, 2020).

The development process in LCDPs is presented in Table 1 and can be divided into five stages. However, the sequence of the stages for some LCDPs can be different because, in some LCDPs, it is better practice to start with the design of the UI since the data model is generated in the background (Sahay, Indamutsa, Di Ruscio, & Pierantonio, 2020).

Table 1: Development stages in LCDPs

Stage	**Description**
Data modelling	Definition of the data model, which represents how the entities used in the application will be structured and connected with each other.
UI design	Design of pages, forms, and other UI elements of the application by using drag and drop.
Business logic definition	Definition of what certain user actions on the UI will trigger.
Integration with other applications	Using services from a third party by API integration.
Deployment	Deployment of the application on-cloud, on-premises etc.

Source: Sahay, Indamutsa, Di Ruscio, & Pierantonio (2020).

The development process in LCDPs is a cycle formed from the stages that end only when the application is built, and its support is done. The flexibility of LCDPs offers users of these platforms to go back and forth through the development stages with a few clicks and easily apply changes to the application.

1.5 Comparison to traditional development

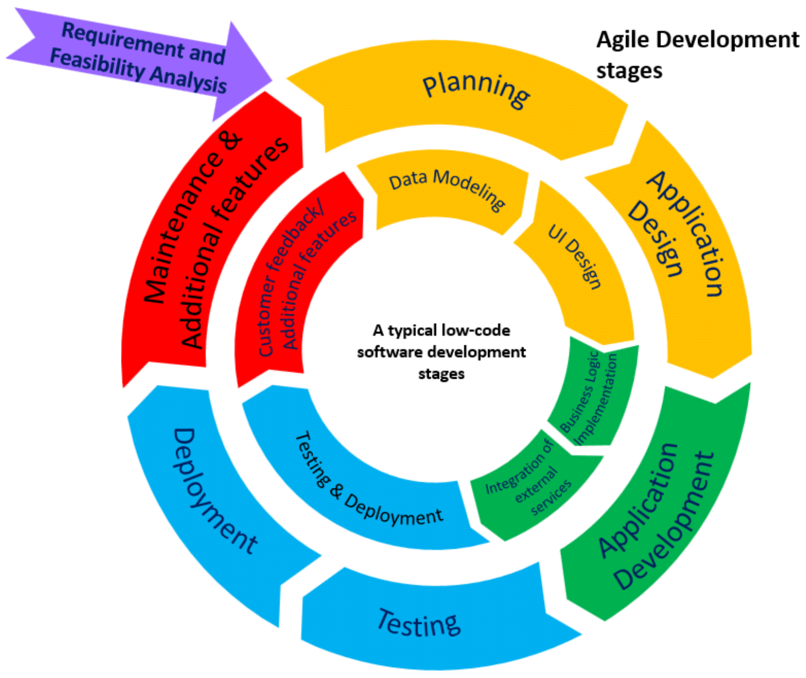
As the LCDPs are becoming more visible on the market, the companies are investing in an analysis of their benefits and downsides compared to their alternatives. The new paradigm for building applications has its positive and negative sides, and each of them should be considered before making the final decision about their adoption.

LCDPs include new members in their target group next to the professional developers: business users with little or no programming skills. This creates an opportunity for companies using LCDPs to include their available business users in the development process and to avoid spending time and money finding additional skilled developers (Mendix, n.d. - g). The non-technical users, called citizen developers, usually stick with the visual development and creation of simple applications. This means that for more complex applications whose requirements exceed the visual drag-and-drop application building, developers are needed (Tozzi, 2021).

In Figure 4, we see displayed the comparison between the traditional and low-code development stages based on the agile methodology. We can see that the development cycle in LCDPs is shorter since some development stages are shorter, and some are merged into one, such as testing and deployment. For example, there is no need to use different software for producing mock-ups of how the UI should look like. That can be done directly in the

LCDP by drag-and-drop capability of offered widgets (Alamin et al., 2021). The reduced time for development is the biggest benefit of using LCDPs instead of traditional development. Because the LCDPs are easy to use and their development cycles are shorter, companies not only save a lot of time intended for development, but they could also save money intended for development costs (Tozzi, 2021).

Figure 4: Comparison between the traditional and low-code development stages based on the agile methodology



Requirement and Feasibility Analysis
Agile Development stages
Planning
Application Design
Application Development
Testing
Deployment
Maintenance & Additional features
Data Modeling
UI Design
Business Logic Implementation
Integration of external services
Testing & Deployment
Customer feedback/ Additional features
A typical low-code software development stages

Source: Alamin et al., (2021).

An additional benefit related to the previous one is the higher productivity. The usage of LCDP can result in higher productivity since some features can be developed and deployed very quickly. The citizen developers can make quick changes to the application through GUI without wasting many resources. Another positive point for the higher productivity of using LCDP is the included collaboration within the platforms. For example, programmers and business people can communicate by writing comments directly on the UI widget, which is more organised and clearer. Additionally, there are fewer repetitive tasks related to problems caused by the usage of different components (Bock & Frank, 2021b).

The LCDPs help businesses to improve communication with clients by providing the tools to create prototypes based on customers' requirements fast and easy. This could be the key to today's challenge of customers' tastes, which are constantly evolving due to the fast-changing market conditions. The prototypes that can be delivered fast by LCDPs can verify if the customer's idea is communicated properly, and estimations on the required work for delivering the actual functionality can be made (Mendix, n.d. -b).

On the other hand, the biggest positive side of traditional development is the opportunity to customise the features of the applications. This is limited if the application is being developed in LCDP because some features cannot be built only by drag and drop and they require writing code. Additionally, only a few LCDPs offer the opportunity to access the source code generated by the platform, which is a big drawback. This means that there will be additional costs if the company wants to switch the vendor for LCDPs since it is dependent on the current one (Alamin et al., 2021).

Companies that use traditional development workflow can develop their solutions in any programming language. However, this is not the case with the LCDPs, which usually support only a few programming languages. The most common programming languages used in LCDPs are Java and JavaScript (Luo, Liang, Wang, Shahin, & Zhan, 2021).

An additional benefit of traditional development is the free choice between a variety of deployment options. Traditional development offers more flexibility when it comes to deployment options since some LCDPs could offer a deployment option that could not align with the client's plan (Tozzi, 2021).

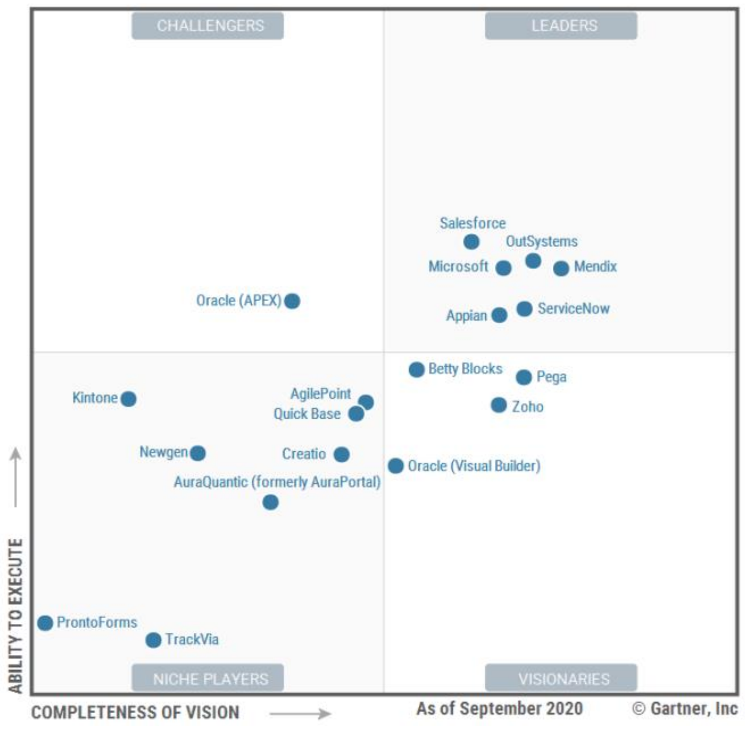
Finally, traditional programming is included even in the LCDPs. It is mainly used when companies are building more complex applications where the components offered by the platforms cannot fulfil the business requirements. In this case, the need for professionals with technical knowledge and experience is inevitable (Tozzi, 2021).

1.6 Market leaders

There are many suppliers of LCDPs on the market that are continually improving their products in the fight to stay at the top of the low-code trend. Gartner (2020) defined the market for enterprise LCDP providers and created a magic quadrant where the top players are presented. This means that these LCDPs are not only intended for individual use but also for meeting the needs on the enterprise level. Since Garner does not consider a significant difference between LCDPs and NCDPs, in their analysis for enterprise LCDPs, NCDPs are included as well.

As presented in Figure 5, there are two dimensions: the ability to execute and the completeness of vision. The most popular and used LCDPs are the ones in the leader's quadrant. Besides the fact that they are pointed out as leaders in the market for LCDPs, they have their own strengths and weaknesses that make them better or worse compared to their rivals (Gartner, 2020).

Figure 5: Magic Quadrant for Enterprise Low-Code Application Platforms



Source: Gartner (2020).

Below we can find the enterprise LCDP leaders based on the current analysis:

The Salesforce platform is ranked as the one with the highest ability to execute. It is a known provider of customer relationship management (CRM) software with strongly established customer relationships. It incorporates various low-code tools that help developers boost their productivity and offers an opportunity for business users to be part of the development. Lightning Flows are one of the many low-code capabilities that Salesforce offers that enable workflow automation (Sego, 2021). Other functionalities include drag-and-drop builders, customisable components, one-click deployment, collaboration between the teams and so on. (Salesforce, 2021).

OutSystems is an LCDP that supports visual development accompanied by Artificial Intelligence (AI) based tools. The applications built by this platform can be cloud-based or on-premises based. The platform offers a wide range of pre-built UI components that are customisable. The product supply is focused on large enterprises for building core systems, internal apps, and customer portals. They have many successful use cases that include CRMs and ERPs as well (OutSystems, n.d.).

Mendix is a leader in the completeness of the vision dimension. It is an LCDP that offers visual development even without writing code. Considering the two groups of users of the LCDP that have different coding experiences, Mendix developed two tools for each group: Mendix Studio for citizen developers and Mendix Studio Pro for developer. Mendix Studio

Pro offers posibility to create more customised solutions by including code. Moreover, the platform offers templates for starting a new application and various pre-built components that can be used in any application (Mendix, n.d. -p).

Microsoft's platform, called Microsoft Power Apps, offers rapid development of business applications. This LCDP offers users to use data from various sources, such as the data stored in SharePoint, Office 356, and other sources. The expression language used in this platform is very similar to writing formulas in Excel, which makes it very easy for citizen developers to learn to use it (Gartner, 2020). Since the platform offers integration with Power BI and Power Automate, users have access to data analysis and easily edit the flow of the application (Heller, 2021).

The Appian platform offers a creation of process-driven applications, where the UI and processes can be built quickly and easily. The platform offers a form-based editor with drag-and-drop functionality and supports the creation of complex logic by using workflows. There is also a possibility for collaboration between different teams, data integration connectors, AI, Robotic Process Automation (RPA) and task management (Gartner, 2020).

ServiceNow App Engine platform offers a collaboration of developers and business users in the same environment. The platform supports a development of web and desktop workflow applications and includes user-friendly UI, a possibility for integrations and process automation, which assure scalability and customisation. Other important features of this platform are chatbots and AI assistance (Torres, 2021).

2 INSURANCE INDUSTRY

2.1 Introduction to the Insurance Sector

Nowadays, the insurance industry can have a significant impact on economic development. The risk management that is offered by insurance companies helps businesses and households to avoid the financial impact of unfortunate events to be handled individually. Mitigation of the losses can have a positive impact on investments, innovation, and competition (Feyen, Lester, & Rocha, 2011).

Insurance companies have a wide range of products that are offered to their clients. The nature of their product is a risk bearing in case of a specific event that could cause financial loss to their clients, and for this purpose, they receive a payment that is called a premium. The premium can be paid at once or many times, depending on the defined payment frequency in the contract (Fabozzi & Drake, 2010). This means that people are cooperating by paying a premium, which is collected by the insurance company and is used to share the risk and financial consequences that could appear if an insured event occurs (Mishra & Mishra, 2016).

In the book “Introduction to Risk Management and Insurance” (1998), the author has two definitions for the term insurance based on its nature. The first one is the financial definition which explains the term as financial compensation for covering a loss that appeared unexpectedly. The funds for compensation of the insured’s losses are called insurance pools, and every insured person contributes to these pools, however, not every person that made a contribution will experience a loss. The premium is calculated by the insurance company based on their prediction of possible losses. This prediction is called exposure to loss, which is another important term in the insurance field. This means that the premium is not the same for every person and event and follows the rule: the higher the expectation of loss, the higher the premium. However, insurance companies cannot predict which individual will suffer a loss and how much this will cost the insurance company. The predictions for loss are basically based on groups of individuals. The second definition is from a legal perspective, and it points out the liability of one party that must make compensation to the party that experienced loss based on a previous agreement. The agreement is called an insurance policy, and the parties included are the insured and the insurer. The insurance policy which represents a contract is regulated by the contract law.

The key concept in the insurance field is risk. A risk is the potential of a harmful event that could happen and cause loss, but it is not known the time and place of occurrence of such an event and if it will happen at all. Other two important concepts are peril and hazard. The first concept refers to an event or condition that could cause a loss or damage while the second concept refers to a factor that raises the probability of peril happening or intensifies the degree of damage that may occur (Rejda, McNamara, & Rabel, 2021).

In the book “Principles of Risk Management and Insurance” (2021), the authors distinguish four main risk categories. In the first category, there are pure and speculative risks that can be distinguished by the result of the event happening, which can be harmful or harmful and beneficial at the same time. The result of the pure risk, if the event occurs, is loss, which means there are no benefits produced by the event. On the other hand, if the risk is speculative, it means that despite the result that could be harmful, there is a possibility of the creation of benefits for other parties. An example of speculative risk is an investment. The second category consists of diversifiable and non-diversifiable risks, whose definitions are mainly based on the scope of the people or groups that are affected by the output of the risk and the chance of diversifying the event outcomes. If the risk does not affect a large group of people and can be diversified, it is called diversifiable risk; otherwise, it is called non-diversifiable risk. In the third group, we have enterprise risk, which consists of many risks that could affect a business’s operations, strategy, and finance. Finally, there are systematic risks that are non-diversifiable and related to the system as a whole. For this type of risk, it is specific that the occurrence of an event in one part of the system, such as one market segment, can negatively affect the whole system and even lead to collapse.

Despite the fact that all of us are exposed to some kind of risk every day, there are some risks that cannot be insured. The risk can be considered as insurable if specific conditions are satisfied, as described in the following section. The risk must be present and important to a large number of people. This is linked directly to the insured’s contributions to the pool of money, and it points out the correlation between the number of insureds and available money to cover the losses. If there are only a few contributors insured against a specific risk, then the premium cannot cover many losses, and to do so, the premium must be very high. An important characteristic of the risks that can be insured is the defensiveness of the caused event in terms of time and space so that the consequences can be properly measured. Uncertainty is another condition that must be fulfilled and refers to the occurrence of the event. Moreover, insurance risks could cause a large loss where the insurance against such risks provides people with financial aid that they could not afford. Finally, the insurance risk must have past statistical data that serves as a base for the calculation of insurance premium and needed pools to cover the losses (Sahoo & Das, 2009).

The insurance sector incorporates three groups of participants:

Group 1. People that use insurance services in order to insure themselves or their assets. In this group, besides people as natural persons, are included businesses as legal persons (Kačar, 2010).

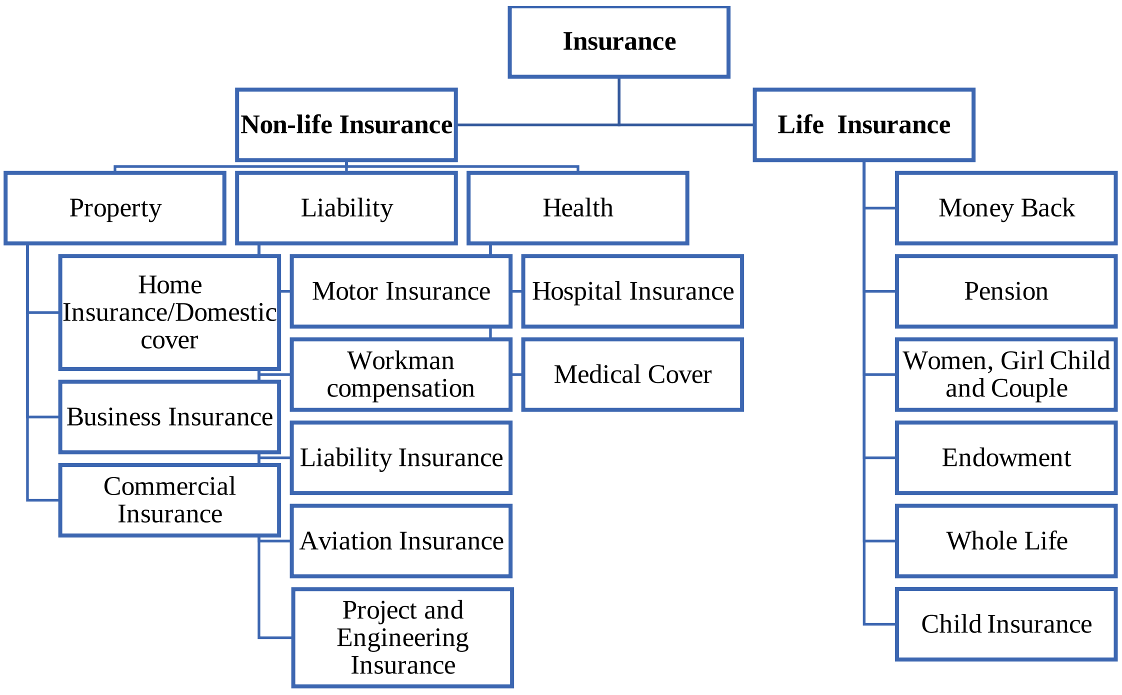
Group 2. Regulators responsible for supervising the insurance process and developing and enforcing rules and regulations in order to enable fair play in the insurance sector. In Slovenia, the main regulator is the Ministry of Finance. Other bodies influencing the insurance market that are part of this group are the Slovenian Insurance Association, Insurance Supervision Agency, and the European Insurance and Occupational Pension Authority (Kačar, 2010).

Group 3. Institutions that offer insurance services. Establishments present in this group include insurance companies, insurance intermediaries and reinsurance companies. Insurance companies are the ones that are legally bonded by the insurance contract to pay compensation to the insured party if an insured event occurs. The insured can conclude an insurance contract directly with the insurance company or through an insurance intermediary, which could be an insurance agent or broker. Insurance agents are people that work in the insurance company or organisations that have signed contracts with the insurance company to promote and sell their products. Insurance brokers are intermediaries that are widely known as experts in the insurance field that can offer clients an insurance product that is most suitable for them. The difference between these two intermediaries is the side of the insurance contractors that they support. The agents support the insurer’s interests, whereas the brokers support the insured’s interests. Moreover, there is another provider of insurance services that have insurance companies as their client. These providers are called reinsurance companies, and their services cover the insufficient funds of insurance companies that are needed to cover the financial loss of their clients. An example of this situation is when there are many claims registered by the policyholder in the same period of time, which usually happens in case of natural disasters (Kačar, 2010).

Marine insurance was considered the earliest form of risk management, and it started with the bottomry bonds. These were loans that were provided by the lenders to the ship owners that were exposed to different types of risks during the transport of their goods. The most common risks were robbery, pirates and bad weather conditions that caused the ships to be drawn. These loans were lent to the boat owners who were obligated to return them with previously agreed interest in case the goods were safely delivered and unfortunate events that could cause loss did not happen. Later, followed other types of insurance such as fire, life, and miscellaneous insurance that were in a different form than today's types (Mishra & Mishra, 2016).

However, the social progress resulted in changes in people's behaviour and needs which led to an expansion of the types of insurance lists. According to Gupta (2008), insurance can be divided into non-life and life branches, as displayed in Figure 6.

Figure 6: Types of life and non-life insurance



Adapted from Gupta (2008).

The non-life insurance group is also known by the name general insurance group and includes financial compensation in case of the occurrence of an event that causes loss that is not death (Acko, 2022). The first group of non-life insurance is the property that includes protection against risks such as fire, marine, theft and burglary. This type of insurance provides protections for property owners that can be natural or legal parties. Property insurance includes home, business, and commercial insurance. Liability is the second group of non-life insurance that includes the risks of damage and injury of a property or person and other liabilities. It covers the costs that emerged because of a harmful events. This group consists of motor, workman compensation, liability, aviation and project and engineering insurance. The last group of non-life insurance is health insurance, where is included protection against two main risks: injury and illness. Here are included hospital and medical cover types (Gupta, 2008). However, the insurance market is rapidly expanding, and new products offered by insurance companies appear on the market. Two of them that are not included in Figure 6 are travel insurance and fire insurance (Acko, 2022).

The life insurance group includes many insurance types that have its mutual goal: ensuring human life. The insured person agrees to pay a premium, and in case of his or her premature death, the beneficiaries stated in the contract receive payment. This means that a certain amount of money at some point in time will be paid back to the insured person in case of period expiration stated in the contract or to his or her beneficiaries, which gives this category of insurance an investment character (Gupta, 2008). The most common types, regarding Gupta (2008), are money back, pension insurance, women, girl, child, and couple insurance, endowment, whole life, and child insurance. Other types of life insurance that are offered by the insurance companies are term life, unit-linked and critical illness insurance (Acko, 2022).

Referring to the annual insurance report overview that analyses the insurance and reinsurance sector of the EEA countries for the year 2021, the biggest part of the life insurance category measured by premium volume by lines of business is taken by Index-linked and Unit-linked insurance with 39%. In second place comes insurance with profit participation at 35%, which is followed by health insurance, which counts 10% of the total premium in the EEA life insurance and reinsurance market. In the non-life category dominates the medical expenses line of business with 18% of total premiums in the insurance and reinsurance market of EEA countries, followed by fire and other damage to property insurance, which counts 17% and motor vehicle liability insurance with 11% (EIOPA, 2022).

2.2 General terms in non-life insurance

In the insurance field, all obligations and rights of the signed parties arise from the insurance policy, which is a document that must contain elements that are determined by the law. In the contract must be insured one or more risks and all information regarding the subject matter of insurance and coverage must be truthfully provided by both parties (Outreville, 1998). However, the insurance process flow does not directly start with the insurance policy. First, the potential insured or the agent must fill in a form that is usually called an insurance application. Then, from the application is created an insurance quote where the entered information can be reviewed, modified, and prepared for the next stage, which is underwriting (Adacta, 2022a). This stage can be automated for some business lines, which means that the company can predefine some rules and if some conditions are satisfied, the quote will be transited to the next status. On the other hand, if any constraint is broken, for example, if the property insured has a value higher than the maximum possible insured amount, the quote must go through the underwriting stage. The person who approves or rejects the quote is the underwriter, who represents the person responsible for assessment of the risk (Gupta, 2008). When the quote is accepted by the underwriter and signed by the customer, an insurance policy is created. In the following sections, the mentioned documents are explained, including the insurance product (Adacta, 2022b).

2.2.1 Insurance product

Insurance companies offer a variety of insurance products, which can be sold in the country where they are based, as well as in other countries. Besides the direct sale to the customer that takes place in the insurance company or online, they can use other sales channels such as intermediaries explained in Chapter 2.1. The common customer is often in a situation where a non-life insurance product is offered to him or her along with the product he or she will buy. This usually happens for products that are part of the electronics or vehicles category (Your Europe, 2022). Insurance products are priced based on a calculation that includes many variables, which are not the same for all product types. For example, for household insurance, one of the variables that affect the premium can be the location where the insured property is positioned (Adacta, 2022c).

2.2.2 Application and Quote

The application is a form that contains the main information about the insured party and their assets that need insurance, as well as information about the needed coverage. For example, an application for motor insurance includes information about the policyholder, insured person, vehicle information and information about the vehicle user, coverages that are requested to be included, payment terms and informative calculation of premium. Regarding the provided information about the coverages, the application can lead to one or more different quotes as the next step in the insurance process (Adacta, 2022a).

An insurance quote represents a document that has all the information provided by the insurance application, including one or more insured objects with information about the calculated premium. If the insurance quote is signed by the client, it means legally binding for the client and the insurance company. For some insurance products, the insurance process does not start with an insurance application but rather with an insurance quote. As mentioned before, the quote must go through the underwriting process before issuing a policy. When the quote is in this stage of the insurance process, the underwriter can approve or reject the quote, and he or she can request additional data from the client or create a counteroffer with changed terms. However, the process flow is not the same for each insurance product and differs based on the country and insurance company as well (Adacta, 2022b).

2.2.3 Policy

The term policy is used for the contract between the insured and the insurer. It states the specific rights and obligations of both contractors. Since many insurance products are offered on the market, some insurance policies are more complex than others. Besides the fact that the structure of the insurance policy is not identical for every insurance product, there is some level of standardisation regarding its structure that points out the most common and necessary elements that must be present in the policy. These elements are (Outreville, 1998):

- Information about the policyholder, insured person, and the insurer
- Information about the insured coverage, including loss events and other important elements, and their clear definition.
- Contract duration
- Limits (for example, about the maximum amount that the insurance company is going to pay to the policyholder in case of a claim).

2.2.4 Property Insurance

Property insurance is part of the non-life insurance group whose object of insurance is a property and its contents. Depending on the risks that will be insured for a particular property, property insurance has a whole palette of products such as fire, home, earthquake insurance, etc. Usually, on the insurance quote, the policyholder can choose the coverage and the perils against which his or her property wants to be insured. However, there is a type of property insurance where the property is insured against any risk except those specified on the exclude list (Sahoo & Das, 2009).

2.2.5 Home Insurance

Home insurance is a part of household insurance that is defined as insurance of a property that the insured owns. It covers the losses that are associated with the insured property and their attached or detached structures that occurred by an event insured and stated in the insurance policy. Home insurance also includes legal liability. The insured objects are buildings, their contents, and outbuilding, depending on the chosen insurance package (Understand Insurance, n.d.). Usually, the insurance of home contents and insurance against catastrophic events such as earthquakes are not included in the basic insurance coverage and could be added additionally to the insurance package (Grace & Klein, 2003). Home insurance policies include a sum insured, which means in case of financial loss, the insurance company will cover the financial loss in the amount of the sum insured, which is stated in the insurance policy. Moreover, there is a type of coverage called total replacement, which means that the insurance company will cover the whole cost amount (Understand Insurance, n.d.).

Home insurance has changed over time regarding the range of risks insured. Multiperil Home insurance appeared in the 1960s, presenting a significant step in the evolution of Home insurance. The present Home insurance product supply on the market results from a history of unwanted events that cause continual modifications of the offered insurance products by the insurers. Higher deductibles for some perils or hazards and risk mitigation credits are an example of modification actions taken by insurers to improve their ability to cover reported claims from their customers (Grace & Klein, 2003).

An example of Home insurance is provided in Section 4.1, where are explained the business requirements for Home insurance based on an analysis of an insurance company offering Home insurance product.

2.3 Digital transformation of insurance companies

Technological development has affected every industry, including the insurance sector. The incorporation of digital technologies by insurance companies was unavoidable, despite the slow-changing nature of the insurance industry. For insurance companies to gain a competitive advantage in the market, they started incorporating new technologies to improve their business models (McKinsey, 2015). Moreover, other events have had a big influence on the insurance companies' success, such as the COVID pandemic. The appearance of the COVID crisis has sped up the digitalisation of some insurance companies. In order to keep the current customers and attract new ones, insurance companies have been closely following tech trends. In such conditions, with many restrictions targeting social contact, the habits of people and companies have changed. The travel restrictions resulted in a lower number of concluded travel policies as well as fewer claims reported by policyholders of travel and motor insurance. On the other hand, the fear of the pandemic resulted in a higher number of concluded life and health insurance policies and a higher number of reported claims in both insurance categories (Bloomberg, 2020).

The new technologies used in the insurance market, bringing significant value for insurers and insureds, are being called InsurTech. The main benefits of the technologies related to the insurance process are more accurate premium calculations, improved fraud detection and improved enhanced techniques for delivering services. The beneficiaries are not only the insurers, but the insureds as well since InsurTech is focusing on improving the experience of both parties (OECD, 2017). There are definitions of InsurTech that classifies it as an ecosystem composed of many components that include parties from different industries (PwC, n.d.). Along with the users of insurance industries, providers such as insurance companies and intermediaries, there are other parties included in the InsurTech ecosystem who use technologies that improve the insurance process. These parties are regulators, institutions from different domains of insurance, such as banks, travel companies, medical providers, and experts in different fields. A very important role in the ecosystem play the InsurTech startups and the BigTech firms that collaborate with the insurance companies to facilitate a digitalisation and improvement of their processes by incorporating InsureTech tools (Volosovych, Zelenitsa, Kondratenko, Szymla, & Mamchur, 2021). A McKinsey analysis shows that the most innovations in the InsurTech field are mainly present in two insurance lines: property and casualty insurance, with 17%, and health insurance with 11%. Regarding the insurance value chain, both insurance lines hold the mentioned percentage in the distribution field, which includes sales (Catlin, Lorenz, Münstermann, & Ricciardi, 2017). Technologies that are commonly used in the insurance sector are mobile technology and applications, AI, Blockchain or distributed ledger technology (DLT), and Smart contacts (OECD, 2017).

The technology of mobiles and mobile applications is used for many purposes such as notification system that insurance companies use to notify the insureds for premium payment related matters. An additional example is the mobile application which is used as a platform where users can register, buy new policies, or access their current ones (OECD, 2017).

AI is another technology that is used by many industries, including insurance. It is intelligence demonstrated by the machines which take actions in order to achieve a desirable goal. An example of its usage in insurance are the chatbots that are based on AI. The potential policyholder can interact with the chatbot and based on the inputs, can get an insurance quote (OECD, 2017). The main enabler of AI usage in the insurance sector is the available data on the history of insurance contracts and claims registered in the system. Moreover, the data gathered from the insured’s smart devices also create a valuable base for implementing AI-based solutions that improve the experience of insurers and insureds. This applies to the Internet of Things (IoT) devices that collect data in the insured’s home. The collected data contributes to more accurate calculation of the premium as well as improved fraud detection (McKinsey, 2021).

Blockchain is a popular technology, especially in the financial services field (OECD, 2017). It is known for removing the need for intermediaries in sharing data such as transactions and, at the same time, ensuring a secure exchange of data within the network between participants. In other words, it can be described as a chain of immutable blocks that contain data (Aloqaily, Otoum, Tseng, & Othman, 2020). The reason that it is listed among the technologies that are often used in the insurance industry is its potential to tackle the challenges that most insurance companies face. Among the most important challenges are market saturation, fraud, inefficient processes such as claims, etc. Moreover, the blockchain has the potential to improve the risk assessment and pricing of insurance products (OECD, 2017). This technology has a big potential for the insurance industry, and yet it is not exploited to a satisfying level. The preconditions of incorporation of blockchain technology are a good understanding of the technology, incorporating other technologies such as AI, advanced analytics, and IoT, and being prepared for making a costly investment that does not yield results in the short term (Shetty et al., 2022).

Smart contacts are closely related to the blockchain since they represent a part of code that is kept in the blockchain, which is automatically executed if a condition is satisfied. The obligations, rewards, and penalties owing to either party of a contract can be stated in the code, which mimics those found in a traditional legal document. An example of smart contracts used in the insurance industry is the occurrence of a catastrophic event such as a flood, where automatic payment of coverage is executed when the damage is validated by gathered data from sensors. (OECD, 2017).

Dealing with the challenge of responding quickly to the changing environment and market conditions, some insurance companies embraced the LCDPs. Even though the usage of LCDPs become more frequent in the insurance market with the appearance of the pandemic, there is evidence that shows LCDPs were used by insurance companies in the past as well (McLaughlin, 2020). Goldberg (2021) mentions the use of Visual Basic as a low-code platform for developing insurance platforms by visual development and minimal coding, the use of Microsoft Access for building databases and SQL language for business rules. However, the author points out the drawbacks of this approach that are related to the maintenance of the developed products since there was no hierarchy of development layers. Moreover, he points out the importance of research that every insurance company has to do before deciding on a low-code development approach in order to create an application that will be easy to maintain and avoid a short-lived life.

Today LCDPs in insurance are used for building mobile and web applications as well as improving some crucial insurance processes such as underwriting, claims and internal reporting. The focus is on customers' needs and experience, so the insurance companies should be prepared for quick responses to the changes in the customer's field to maximise their value. The changes could be required in business processes, flows or UI of their platforms, which are the most important communication channel with their customers. Moreover, there are low-code and no-code development tools for a chatbots that enable businesses to create a chatbot without coding in order to communicate with their customers. The chatbots can be integrated with other platforms, such as Messenger or can be added as a widget to their official site (Shakeel, 2022).

The power in the insurance field comes from the available data and the ability to analyse it and discover correlations that could bring a huge advantage to insurance companies. Insurance companies do not always have direct contact with their customer since there are insurance intermediaries. In order to improve their market positions, they need to know their customers, which can be realised by digitalisation and incorporation of different tools. However, it is also important to mention that digitalisation also brings risks as well, such as the risk from hackers (McKinsey, 2017).

3 OVERVIEW OF THE ANALYSED LOW-CODE PLATFORMS

3.1 AdInsure

AdInsure is an end-to-end platform that is used in the insurance field. Its main characteristics are high agility and accelerating innovations since it has an open architecture and a low-code IDE tool called AdInsure Studio that makes it easy for insurance companies to digitalise their products, configure features tailored to their needs and connect with their stakeholders (Adacta, 2022e).

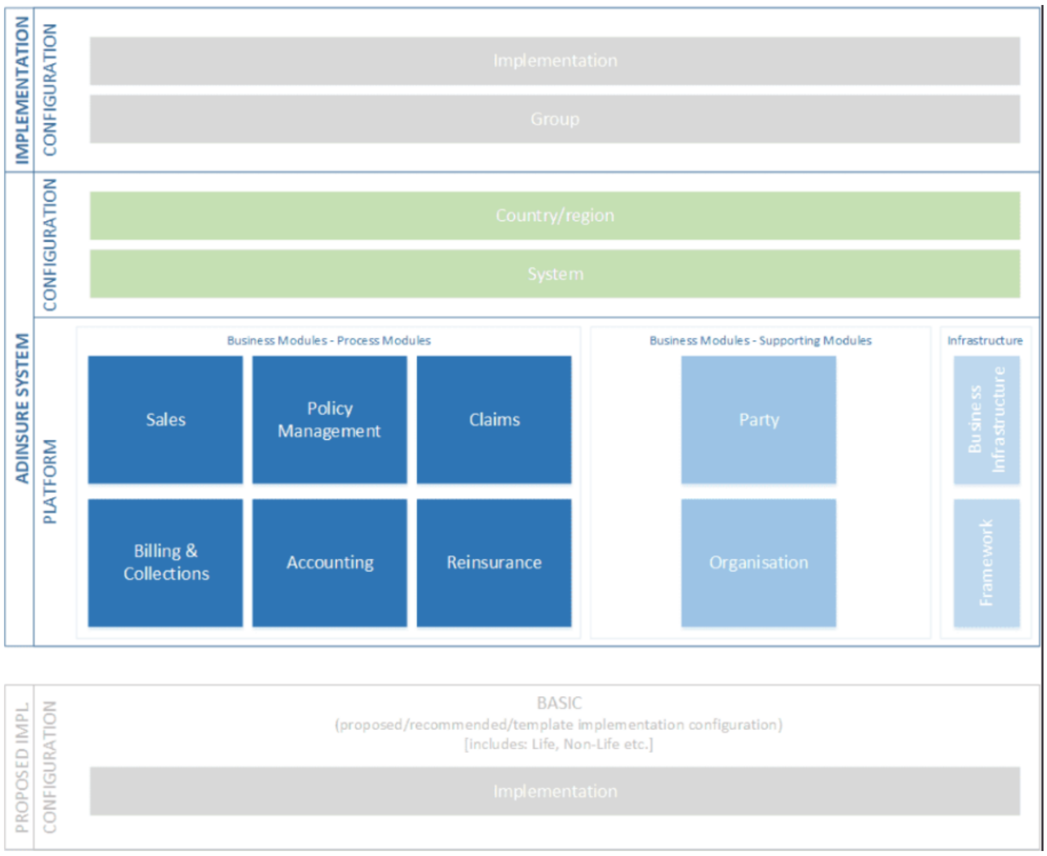
3.1.1 Company background

The AdInsure platform is created by the software company Adacta which provides services in the insurance industry, which include the development and implementation of IT solutions and consulting. The company was established in 1989 and its headquarters are located in Ljubljana, Slovenia. The main shareholder is Volpi Capital. In addition to the existing headquarters in Ljubljana, there are six offices located throughout Europe: Croatia (Zagreb), Serbia (Belgrade), Czech Republic (Brno), Cyprus (Nicosia), the Netherlands (Amsterdam) and another one in Slovenia (Maribor). Adacta counts more than 350 employees spread among different offices in Europe, and it has a goal of broadening its borders. Its presence on the market for more than 30 years has contributed to successfully concluding many projects for insurance companies, including more than 20 implementations of their IT solutions. Moreover, its successful path witnessed the attention of successful research companies such as Gartner, which included Adacta in its magic quadrant as a niche player for non-life insurers in Europe (Adacta, 2022e).

3.1.2 AdInsure Platform

AdInsure consists of two main parts: AdInsure platform framework and services and configuration. Its structure is presented in Figure 7. The platform part consists of three main parts: process business modules, supporting business modules and infrastructure. On the bottom are placed the framework and business infrastructure. The framework contains the definition of all insurance-specific entities, such as documents and master entities. For example, here is defined the structure of the document, such as the insurance quote. Therefore, if the configurator does not follow the structure defined in the platform, the document will be invalid and cannot be published. The business infrastructure includes the management of activities, printouts, attachments, and users. An example of activity management is an action that is required from a user to be executed on a document, such as confirmation of a constraint on a quote that is in status “In Underwriting”. This activity has to be executed by a user that has an underwriter role, so the quote can transit to the next state. Above the infrastructure part are placed supporting business modules, such as party and organisation modules. These modules have two usages: they can be used as standalone modules and as support to the business modules. They contain functionalities that are shared between the other modules. At the top of the platform are placed the process modules. These types of modules are related to the business and are concentrated in one business area. Every module has defined APIs, and there are three types: internal, shared, and public APIs. The modules communicate through shared APIs. For example: when the policy management client component searches for parties, it accesses the party module through shared APIs (Adacta, 2022d).

Figure 7: Composition of AdInsure platform



Source: Adacta (2022d).

The configuration part is a set of business functionalities that are positioned in different layers. The packages in the standard (country/region) and system layer are, by default, included in the software. The users have the ability to install any of the system and standard packages that they need. The packages include a basic configuration of the business modules, such as Sales, Policy Management, Claims, Billing and Collections, Accounting, Reinsurance, Party and Organisation. In configuration can be used predefined platform element types which can be composed of configuration elements. An example of it is a configuration of the master entity platform element called contract type, which contains the possible contract types: application, quote, and policy. The master entity is a platform element, and the configuration items that can be added to the master entity are general properties, data schema, validations, client actions, UI schema, translations, attachments, and mapping (Adacta, 2022d).

In the implementation layer is positioned configuration specific for an insurance company. Insurance companies could operate in the same market segment, such as life insurance, however, their products do differ. They have the ability to use the configuration defined in the lower layer that is by default included in AdInsure and upgrade it by configuring their own products and processes. For example, the insurance company could have a specific type of contract that is not included in the contract type master entity provided by the system package. Since the implementation teams cannot change the system and standard layers, they must override the existing master entity contract type in their layer and add their specific contract types (Adacta, 2022d).

The approach of overriding configurations in higher layer points out one of the top features of AdInsure, which is extendibility. Moreover, reusability is another feature that is part of this group. The configurations in the system and standard layer are organised into components that can be reused in many different configurations. Extending and reusing platform elements can be achieved with a few clicks in the AdInsure Studio (Adacta, 2022g).

User, configurator, and external application are the three groups of users of the AdInsure system. The first group includes business users that use the business functionalities of AdInsure. The second group includes developers or business users that, by using the AdInsure Studio, add or change new business functionalities of the system. In the third group are included all systems that access the AdInsure functionalities through APIs (Adacta, 2022f)

3.1.3 AdInsure Studio

AdInsure Studio is a low-code tool for AdInsure Platform that is necessary for customising the platform elements. It is an extension that can be added to Visual Studio Code, and it represents an intuitive IDE. It offers the ability to produce new configurations through wizards and generators as well as edit the existing configuration with user-friendly GUI editors. Additionally, AdInsure Studio supports publishing and deployment of the produced or changed configuration. AdInsure can be deployed on the cloud in AWS and Azure, or it can be deployed on-premises.

In the AdInsure Studio are included wizards for generating life and non-life insurance and sales products. The biggest value of the wizard is the possibility of producing a sales product or insurance product within a few steps. Insurance product and sales product are terms used in AdInsure that refer to a set of configurations of various rules that define a present or future product that will be offered by the insurance company. The insurance product contains configurations such as insured object types, coverage attributes, underwriting rules, payer rules, premium calculation, and function helpers. It is closely related to the term of the tariff, that in AdInsure glossary refers to the evaluation of the contracts such as applications, quotes, and policies. The sales product contains configurations about the document (for example, Home insurance quote), including UI definition and other elements used on the document such as client actions, data sources, data providers etc. Sales products can use one or more insurance products, depending on the insurance type. They use an already prepared configuration from standard and system layers, such as components for UI parts and libraries for client actions which are generated based on the user's answers in the wizard. However, the produced files, such as business rules, UI schemas, and data schemas, must be edited by the user to apply product specification. In other words, the wizards produce the basic configuration, which can be edited by business users or developers by using user-friendly editors. This configuration is placed in the implementation layer, which is placed above the system and standard layers (Adacta, 2022h).

There are many different editors that are available within the AdInsure Studio, such as general properties editor, workflow editor, form editor, data model editor and rule editor. Their functionality is described later in this chapter, where I used them to create a Home insurance product. Despite the mentioned functionalities, the AdInsure Studio provides many types of explorers (configuration, environment, and gallery explorer), the ability to write tests for local testing, multi-language support of configuration items and support of scripts that can be executed on Continuous Integration (CI) (AdInsure, 2022i).

Based on their technical experience, the user of AdInsure Studio can be grouped into two groups: users with technical experience, which includes developers and tech-savvy business users and users with no technical experience, which includes business users. Since the generated code by the AdInsure Studio can be accessed and modified, there are two available modes for modifying configuration: basic mode and advanced mode. In basic mode, the configuration is opened by the AdInsure editors and is modified there, for example, by using the drag-and-drop principle or filling a table that presents a business rule. All configurations can also be opened in advanced mode as a form of generated JSON and JavaScript files.

3.2 Mendix

Mendix is a leading platform for the creation of web and mobile applications on the LCDP and NCDP market for enterprises. This fact is also confirmed by its leading position in 2020 in Gartner Magic Quadrant and Forrester Wave. It has two dedicated IDEs for developing applications for each group of users: Mendix Studio for citizen developers and Mendix Studio Pro for developers. Today it counts more than 300,000 developers and more than 50 million users (Mendix, n.d. -a).

3.2.1 Company background

Mendix was founded in the Netherlands in 2005. The idea of its creation was based on an issue related to communication between business users and developers. The issue with understanding the business requirements was overcome by the introduction of visual development based on the MDD principle, which created plenty of benefits for both sides: the developers and the business users (Mendix, n.d. -b). Its headquarters are based in Rotterdam, the Netherlands. Despite the three offices in Europe (the Netherlands, UK, and France), Mendix offices are present all around the world: North America (Canada, United States), MESA (UAE, South Africa), Asia (China, Hong Kong, India) and Australia. The company grew in the next years by selling its main product (Mendix, n.d. -d) - a successful platform that attracted many investors such as Battery Ventures, Prime Ventures and HenQ. In 2018 Mendix was acquired by Siemens, which is a company that operates in the fields of energy, transportation, healthcare, etc (Crunchbase, n.d.). Behind the well-known brand of Siemens there are three companies, and each of them is concentrated on a particular sector. These companies are called Siemens AG, Siemens Energy and Siemens Healthineers (Siemens, n.d.).

3.2.2 Mendix Platform

The Mendix platform consists of three main parts: Developer Portal, Mendix Studio and Mendix Studio Pro. The Development Portal offers an overview of all created applications by the user and other users that are part of the same company (Mendix, 2023a). Moreover, there is enabled collaboration between the team members and management of different projects by application of Agile methodology which is based on Scrum and Kanban approaches. The users can track all activities related to a shared project and can communicate with their team members through the comment section. Users can have different access right for a project based on their application role. For example, Scrum Master, which is a role defined in Scrum methodology, has the right to add or remove tasks from the current Sprint (Mendix, 2023c). An additional feature that is part of the Developer Portal is version control. It is enabled by the Team Server, which is a plug-in of the Development portal, where are stored all applications versions that have been committed. The changes in the applications that are committed can be directly linked to user stories present on the Scrum board, and this is enabled by the Team Server as well (Mendix, 2023a).

The other two parts of the Mendix platform are Mendix Studio and Mendix Studio Pro, which are low-code IDEs used for application development. A description of their purpose and main functionalities can be found in Chapter 3.2.3.

Mendix represents an ecosystem that, besides the mentioned main parts of the platform, it includes other parts as well: Marketplace, Atlas UI, Data Hub, Support, Community and Academy (Mendix, 2023a).

In the Marketplace are listed reusable components that the users can download and use in their applications. These components could be modules, widgets, and other features that the developers and citizen developers use with the purpose of building an application. Despite the components, there are environments and applications ready to download that are available to anyone or for a particular group of people that work for the same company. The users can search for needed components for their applications and filter them by industry, content type, compatibility, tags, and rating. Additionally, the users can share the content that they have created with other people (Mendix, 2023a).

Atlas is a UI framework which offers a variety of reusable UI components, templates, and themes with customisable behaviour. It is a cross-platform framework, which means that it is compatible with web, native mobile and Progressive Web Apps (PWA). The Mendix platform includes the Atlas UI framework, and its functionalities can be used through the two Mendix IDEs (Mendix, n.d. -f).

Mendix Datahub is another component of the Mendix ecosystem that represents a central hub that contains datasets from different data sources, which connection is made based on their metadata. Despite sharing data between the applications developed in Mendix, the Mendix Datahub enables connection to applications that were not built in Mendix (Mendix, n.d. -e).

The Support component refers to the support team with whom the users can connect and report issues. The Community component contains channels where the users connect, inspire, and help each other. This includes Mendix forum, blogs, documentation and other user with whom the user can connect. If the issues are related to the application development, the user gets support from other Mendix users on the Mendix forum. It is a place where the users can post their questions, ideas and answer other user’s questions. Additionally, Mendix engages its users to contribute to the Mendix forum, documentation, training, and other activities by rewarding them with points, badges, and credits. There are many leaderboards based on different activities that rank the top users (Mendix, n.d. -g).

The last component of the Mendix ecosystem is Academy, where the user can sign up for different courses based on their technical knowledge. Moreover, there is a section with learning paths on different topics and levels that help users to broaden their knowledge about Mendix (Mendix, n.d. -e).

Mendix has many successful customer stories from different industries, such as banking, insurance, education, energy and utilities, logistics, retail etc. Despite them, many templates for a specific industry are available on the Mendix Marketplace that can be downloaded and tested for free or used as base configuration. For insurance, Mendix has concentrated on three key areas: distribution, claims and underwriting. In each area Mendix has offered solution on the Marketplace with a description of the key functionalities of each of them. An example of an application created in Mendix for insurance companies is FaceQuote, that incorporates AI and Machine Learning (ML). The customer needs to take a selfie and upload it through the application. The goal of the application is to offer a premium estimation to the customer based on the user age analysis of the photo (Mendix, n.d. -h).

3.2.3 Mendix Studio and Mendix Studio Pro

Mendix Studio and Mendix Studio Pro are IDEs used for the Mendix platform. The differences between these two IDEs are based on the user and their technical experience. The first one is used by business users or so-called citizen developers, who can build an app without writing any code. It is a “What you see is what you get” (WYSIWYG) editor, also known as a web modeler, which enables application building by dragging and dropping elements that are provided by Mendix. These elements can be related to UI (building blocks and widgets), domain model or application flow (Mendix, n.d. -e).

On the other hand, Mendix Studio Pro is known as a Desktop modeller and its users are developers and business people that have experience in application development. It offers more features than Mendix Studio, which enable the building of application by using ready-to-use components, creating customised components and extending the application by using JavaScript, Java, and CSS. Moreover, integration with other applications is also possible through Mendix Studio Pro (Mendix, n.d. -i).

The UI of the application is modelled in the Page editor, where the user can define the look of a page by choosing a page layout, template, drag-and-drop elements on each page and add events. The drag-and-drop elements are called widgets and an example of it is a check box which belongs to the input elements category. The application logic can be configured visually in three ways: microflow, nanoflow and workflow. For modelling application behaviour Mendix uses Business Process Model and Notation standardisation (BPMN). The data architecture of the application can be modelled in the Domain model that represents all entities and their relations. During application development, the developers have support from the Mendix platform through its bots that are driven by AI and ML. This is enabled through MxAssist Logic Bot, MxAssist Performance Bot, and Validation Assist. A detailed description of the mentioned editors and functionalities is given in Chapter 4 where is described the implementation of the insurance product (Mendix, n.d. -e).

Mendix Studio or Mendix Studio Pro are the main modellers in Mendix. One of the great benefits of using these modellers is improved communication since the models can be quickly created and effectively communicated with the stakeholders (Mendix, n.d. -b).

4 IMPLEMENTATION OF HOME INSURANCE PRODUCT IN ADINSURE STUDIO AND MENDIX STUDIO PRO

4.1 Business requirements

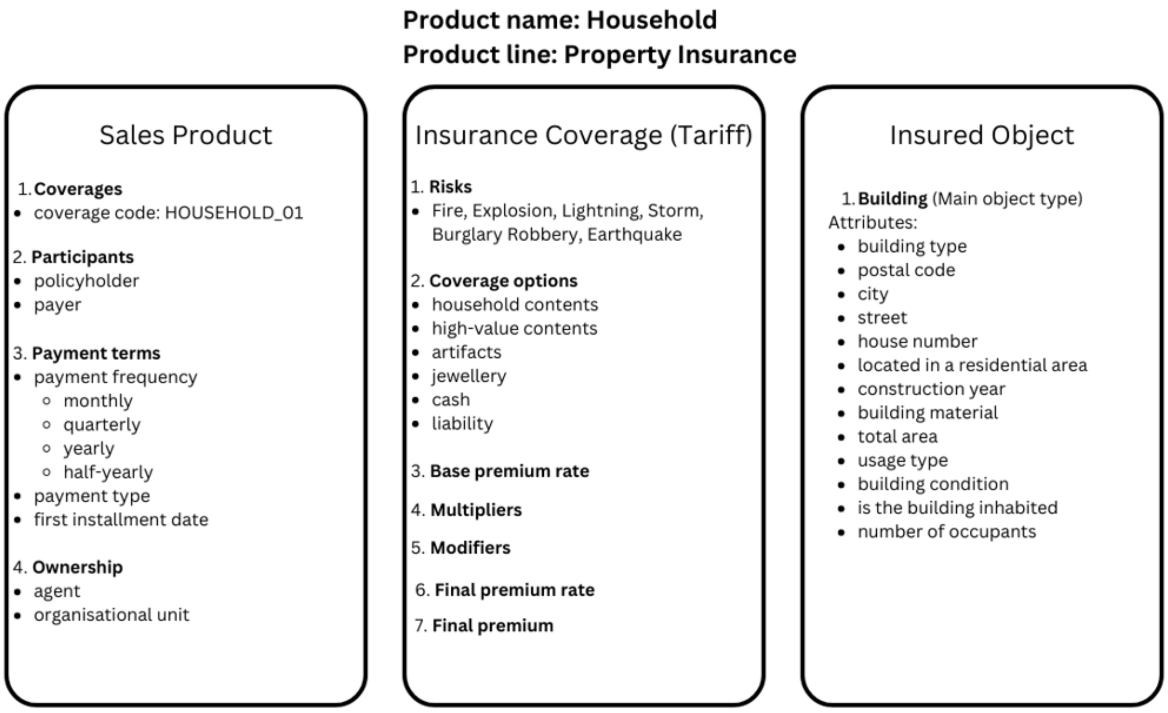
Analysing business processes and defining business requirements are important stages of every software development cycle. Depending on the methodology, the analysis of business processes and definition of business requirements can be done only in the first stage of the development cycle (e.g., waterfall methodology), or analysis could be done as continuous activity (e.g., agile development process). The waterfall methodology is a traditional software development methodology that has a linear development process, which means when one phase is completed, the next phase can start. Regarding the business requirements, this means that the analysis of business processes is done in one phase and then begins the next phase, which is design. Changes in requirements in later phases are unwanted, and they result in higher costs. In contrast to that, agile development concentrates on short development cycles and frequent deliveries of software. Since the development process is not linear and all activities are continuous, the client could change or add new requirements in later phases of the software development (Waja, Shah, & Nanavati, 2021).

Business analyses are done by business people by gathering information from clients about their business processes and needs, analysing and translating them into user requirements that serve as guidance for the development team. This is usually done by business analysts. The analysis could take several forms: interviews, questionnaires, meetings etc. The quality of analysis has a big impact on the project duration and costs, so it is very important to have clear communication with the client. After business analysis, the business requirements are defined and communicated to the development team as user stories, tasks, etc. Miscommunication with the client could lead to the development of wrong features and a waste of valuable recourses (Paetsch, Eberlein, & Maurer, 2007).

Adacta uses Scrum methodology, which belongs to agile software development. For agile methodologies is typical a small number of documented requirements at the beginning of the project since the focus is on the delivery of most needed features, and the details are left for later (Paetsch, Eberlein, & Maurer, 2007).

In Figure 8 is presented an example of business requirements for a Home insurance product. Since this data is gathered from a real insurance company that is currently active on the market and its data is confidential, changes to each part of the defined requirements were made. For this analysis was sent a questionnaire to the client with standard questions for getting a rough picture of the business process of the insurance company. Additionally, several meetings with the client were organised to get more details about their processes. The result of the analysis was a definition of three documents where were defined insurance coverage, insured object, and sales product. The business requirements which were used for the development of the scenario are attached as Appendix 2.

Figure 8: Business requirements for Home insurance



Product name: Household
Product line: Property Insurance

Sales Product

1. **Coverages**
- coverage code: HOUSEHOLD_01

2. **Participants**
- policyholder
- payer

3. **Payment terms**
- payment frequency
  - monthly
  - quarterly
  - yearly
  - half-yearly
- payment type
- first installment date

4. **Ownership**
- agent
- organisational unit

Insurance Coverage (Tariff)

1. **Risks**
- Fire, Explosion, Lightning, Storm, Burglary Robbery, Earthquake

2. **Coverage options**
- household contents
- high-value contents
- artifacts
- jewellery
- cash
- liability

3. **Base premium rate**

4. **Multipliers**

5. **Modifiers**

6. **Final premium rate**

7. **Final premium**

Insured Object

1. **Building** (Main object type)

Attributes:
- building type
- postal code
- city
- street
- house number
- located in a residential area
- construction year
- building material
- total area
- usage type
- building condition
- is the building inhabited
- number of occupants

Source: See Appendix 2.

The business requirements for Home insurance are divided into three parts: insurance coverage, insured object and sales product.

The insurance coverage part contains general data about the coverage – tariff. It is a set of business rules that determine how the total premium is calculated. The content is divided into seven parts: risks, coverage options, base premium rate, multipliers, modifiers, final premium rate, and final premium. The risk part contains the set of risks covered by the insurance policy. In coverage options are listed all types of insurance coverage that are available. Here is also defined the business rule of calculating the sum insured for each selected coverage option. The base premium, which is used for the calculation of the total premium, differs based on the chosen building type. Moreover, in this part are defined business rules, multipliers, and modifiers. Multipliers are determined by the user input about the main object, which is the building. They can decrease or increase the premium rate. Every coverage has its own modifier that is fixed and used in the premium rate calculation. The final premium rate is a business rule that calculates rates for each coverage option. Premium by coverage is calculated by multiplying the calculated rate with the sum insured per coverage option. The sum of all calculated premiums results in total premium.

The insured object document contains data about the types of insured objects and insured objects' attributes. Currently, there is one object type that also represents the main object type: Building. For each attribute is specified the logical name used in the system, attribute type, default value (if it exists) and if the attribute is mandatory. The specified data is filled in by the user directly on the quote.

Finally, the sales product document contains information about the main settings of the sales product, the available coverages, participants, payment terms and ownership. Main settings relate to general sales product configuration, such as product name, product code, numbering prefix and date from which this configuration is valid. For this product is available only one type of coverage (Household) and its code is specified in the coverages section. Participants are all parties stated on the insurance contract. They are policyholder and payer and are mandatory to be specified on each insurance contract. The policyholder is considered an insured person. Payment terms contain information on available types of payment frequency and payment modes that need to be specified by the user on the insurance quote. Ownership is the last part of this document that defines the needed input from the user about the agent.

The expected result of this scenario is the ability of the agent to create a Home insurance quote in order to prepare an offer to his or her client. There are two main users: agent and customer. The agent should be able to define all data about the policyholder, contract duration, insured object, and ownership. When the data is prepared, the quote should be issued by the agent and ready to be reviewed by the customer. If the customer accepts the offer, the quote should be signed, otherwise, it should be rejected. There should be also a user with an administrator role that will have permissions as a customer and agent.

The mentioned scenario is limited to the creation of insurance quotes. In real life, there is an underwriting process where the quote could be rejected or accepted based on the evaluation of the property and policyholder. Moreover, the next stage after signing the quote by the customer is the creation of a policy.

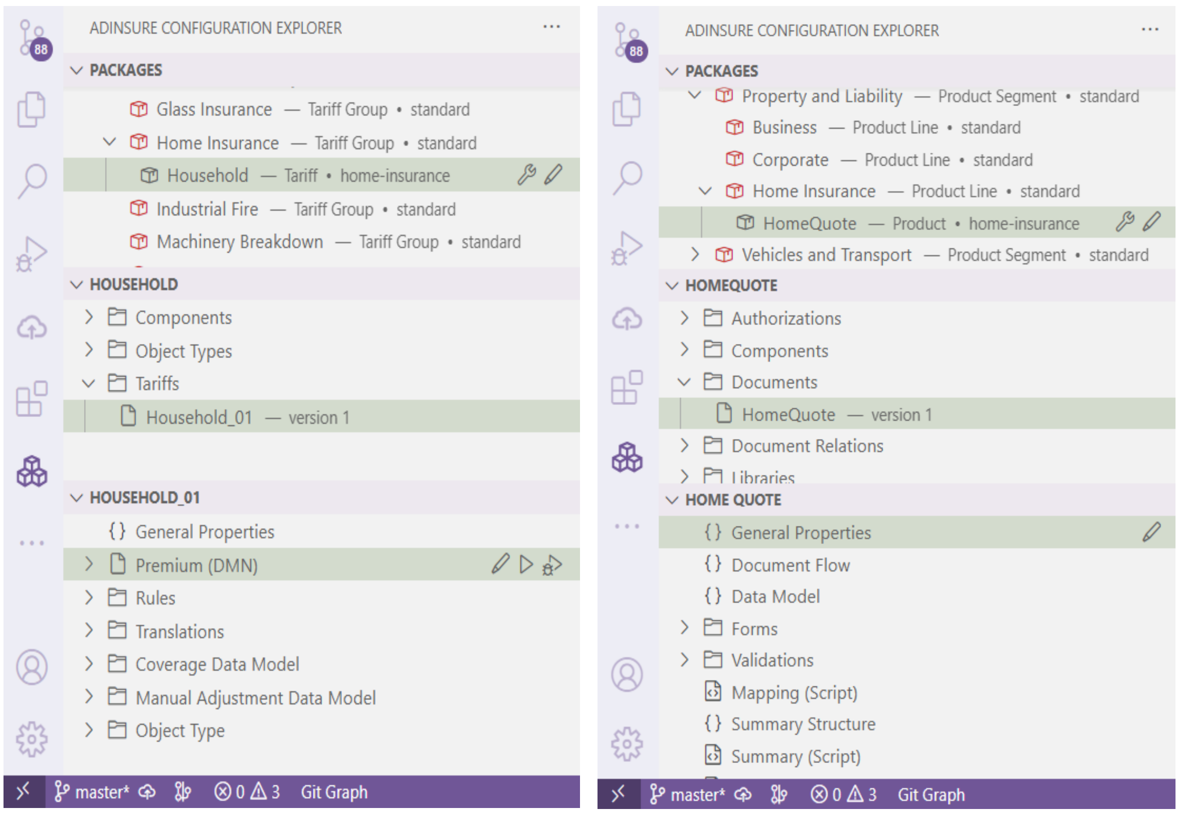
4.2 Configuration preconditions

4.2.1 AdInsure

As mentioned in Section 3.1.3, AdInsure offers wizards that generate configuration files for insurance and sales products. The insurance product can be defined by using the non-life insurance product wizard, where the user have to answer predefined questions. The questions are grouped in several steps, depending on the product type. The wizard for the creation of a non-life insurance product has the following steps: general properties, object type, object subtype, coverage attributes, required rules, underwriting and payment terms. When the user answers all questions, a configuration that corresponds to the user's choices is generated. After the generation of the configuration, the user can define the premium calculation, limits, deductibles, and other important details. Moreover, if the user forgot to add some parts by the wizard, they can always be added after the execution of the wizard. In AdInsure, there are some predefined rules that are valid for the produced product by the wizard. However, if the generated product has some characteristics that do not apply to the predefined logic, the users must define this logic by themselves. These rules could apply to invoicing, currency, tax rules etc.

I began with the creation of a new workspace called Home insurance. In the new workspace can be added packages from the system and standard layers which contain all the needed configurations to start off a new project. This can be done by installing packages from AdInsure Gallery Explorer, where are listed all available packages. With the creation of the project, there was a new layer added which was placed above the standard and system. I used the non-life insurance wizard to create a Home insurance tariff. When I answered all questions in the wizard, a message in the output console was displayed that the wizard was executed successfully, and a new folder was generated, as displayed in Figure 9 (left).

Figure 9: Generated Household tariff and Home sales product in AdInsure Studio


ADINSURE CONFIGURATION EXPLORER
PACKAGES
Glass Insurance — Tariff Group • standard
Home Insurance — Tariff Group • standard
Household — Tariff • home-insurance
Industrial Fire — Tariff Group • standard
Machinery Breakdown — Tariff Group • standard
HOUSEHOLD
Components
Object Types
Tariffs
Household_01 — version 1
HOUSEHOLD_01
General Properties
Premium (DMN)
Rules
Translations
Coverage Data Model
Manual Adjustment Data Model
Object Type
master* 0 3 Git Graph

ADINSURE CONFIGURATION EXPLORER
PACKAGES
Property and Liability — Product Segment • standard
Business — Product Line • standard
Corporate — Product Line • standard
Home Insurance — Product Line • standard
HomeQuote — Product • home-insurance
Vehicles and Transport — Product Segment • standard
HOMEQUOTE
Authorizations
Components
Documents
HomeQuote — version 1
Document Relations
Libraries
HOME QUOTE
General Properties
Document Flow
Data Model
Forms
Validations
Mapping (Script)
Summary Structure
Summary (Script)
master* 0 3 Git Graph

Source: Own work.

When I finished with the configuration of the Home insurance product, I created a new sales product called Home Quote by using the non-life sales product wizard. This wizard has the same functionalities as the previous one, and the difference is that this wizard generates sales product specific configuration. Additionally, in the second step, the user must select a tariff that will be used for this sales product. I selected the Household tariff. The generated files are displayed in Figure 9 (right).

AdInsure Studio displays a warning to users that they must enter data in the generated rules. The configuration of the rules is explained in the following sections.

4.2.2 Mendix

In Mendix, I started with the creation of a blank application on the Developer portal. Mendix offers many application templates with feature descriptions of included functionalities and examples that can be used as starting point for building applications. For the insurance industry, there was a Claims template available; however, the user could create a new template that could be used for the creation of the following projects. After the application creation, Project Buzz is displayed, which represents a board of project activities, and its main purpose is a collaboration with the company members that have access to the created project. Then, I opened the project in Mendix Studio Pro, where I continued with the development of the Home insurance product. All projects contain some default settings and modules provided by Mendix that can be changed by the user. In the created module, there are generated empty domain model, a page, and a collection of images, where the user can add images that could be used on UI, for example, as button icons.

4.3 Data modelling

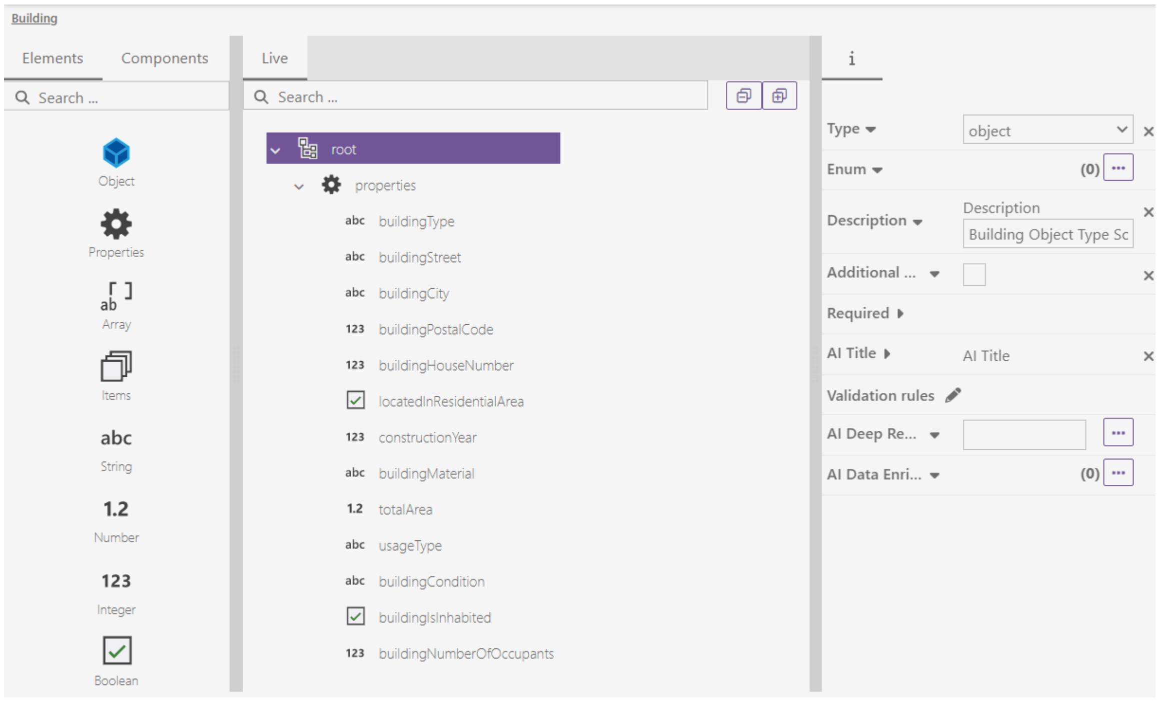
4.3.1 AdInsure

The data model in AdInsure Studio looks different than in Mendix Studio Pro. The AdInsure Platform is constructed in a way that every component must have its own data model. Considering the platform requirement, every component had its own data schema. The attributes can be added by the AdInsure Studio, and the user has the freedom to define them without any restrictions. The components can be used on different documents and are usually created for configurations that are used on many places to prevent duplication of configuration. For example, data about the policyholder, such as name and address, is required on every quote. Because of that, there is a component that contains these fields and is referenced on many quotes.

However, there are many restrictions defined on the platform level. These restrictions are related to the insurance nature and the user must follow them in order to have a working insurance or sales product. For example, the user must define specific properties to all sales and insure products. The defined properties are part of the evaluation process of the insurance product, and its logic is provided by the platform, which is described in Section 4.5.

Figure 10 displays the data model of the Building component. The user can choose elements or components from the left grid and add them to the middle grid to create a data model. On the right grid can be specified rules for each property, such as validation rules and value restrictions. As mentioned, this is a component which means it can be used in many places in the same or different layers. Despite the Building component that was created for the main object type, with the help of the non-life product wizard was generated another component called Coverage Options, which contained properties regarding the coverage options. AdInsure uses a relational database, and the inputs from the users are saved in tables in a predefined structure by the platform.

Figure 10: Household coverage component in AdInsure Studio – data model



root
properties
abc buildingType
abc buildingStreet
abc buildingCity
123 buildingPostalCode
123 buildingHouseNumber
✓ locatedInResidentialArea
123 constructionYear
abc buildingMaterial
1.2 totalArea
abc usageType
abc buildingCondition
✓ buildingIsInhabited
123 buildingNumberOfOccupants

Type	object
Enum	(0)
Description	Description: Building Object Type Sc
Additional ...	
Required	
AI Title	AI Title
Validation rules	
AI Deep Re...	
AI Data Enri...	(0)

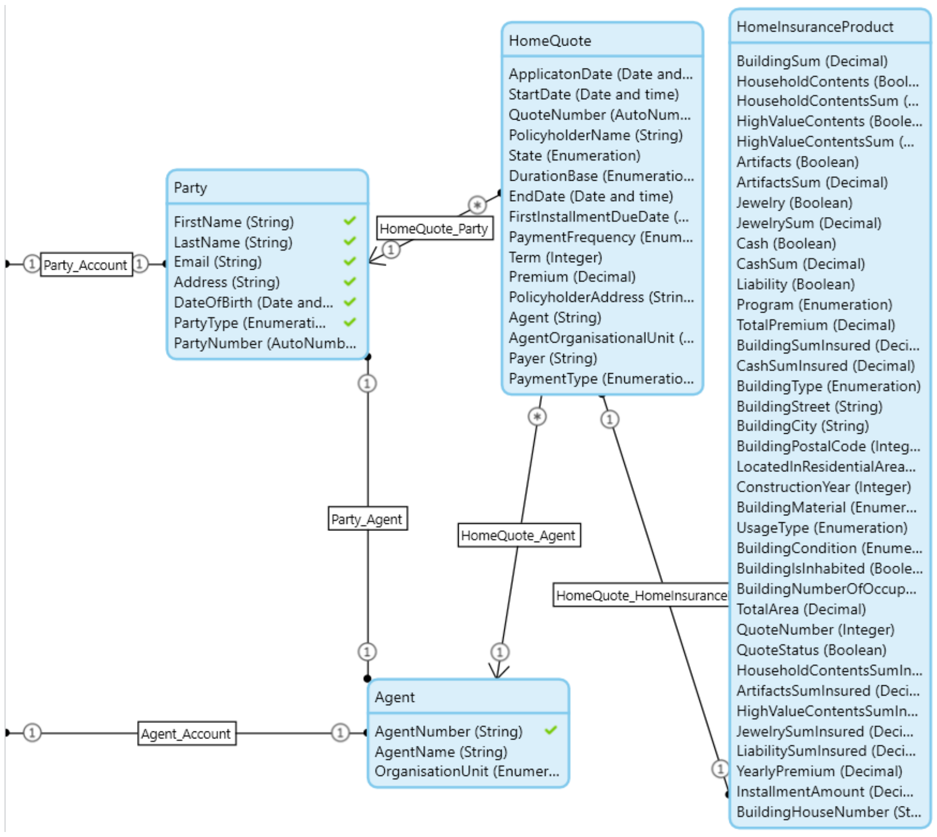
Source: Own work.

4.3.2 Mendix

In Mendix, the data model is called domain model, and the user can there define the entities and their attributes. Each module should have its own data model. Every entity defined in the data model is created as a table in the database, and every attribute inside an entity is defined as a column inside the table (Mendix, 2022b). Each entity can be connected to another entity through association. Regarding the defined business requirements, there were four needed entities in Mendix: agent, party, Home insurance product and home quote.

Figure 11 presents the data model in Mendix Studio Pro. Each entity with its attributes represents an object stored in the database. For example, the party entity is a blueprint of the parties included in the insurance contract. This could be an insured person or an agent. Party has a one-on-one association with the Agent entity, which means one party can be registered as one agent. The reason that these two tables are separate despite their one-on-one association is because in reality the business scenarios are more complex, and more service providers are included. The same reason applies for other tables that have one-on-one relations. The association with the Home Quote entity is one-to-many, which means that one party can have multiple quotes. The same type of association is present between the Agent and the Home Quote entity, which means that an agent can create multiple quotes. Since the Home insurance product in the described scenario can have one insured object, the relation between the Home Quote and Home insurance Product is one-on-one. The associations Party_Account and Agent_Account lead outside the Property Insurance module and connect Party and Agent entities with the Account entity from the Administration module that was added from Mendix Marketplace. These cross-module associations point out that a party and agent can have only one account as a user in the application.

Figure 11: Data model in Mendix Studio Pro



Source: Own work.

Changes in entities can be frequently made in the domain model. The existing attributes can be used in many microflows, however, changes of these attributes in the domain model won't cause any issue in the application since Medix will apply these changes in each place the particular attribute was used. Therefore, the user does not need to care about synchronising of changes in the database since this is automatically done on publish of the application. In each entity can be defined rules that apply to the whole entity or separate attributes such as validation rules, access rules, event handlers etc.

4.4 User interface

After the definition of the data model, the next step is the design of the UI form where inputs are entered by the user. The goal of every application is to have an intuitive UI, which means the user understands the application's behaviour and doesn't need time to think about what have to be clicked or entered. On the UI must be present the properties and attributes defined in the data model as inputs required from the user. Not all properties and attributes are required inputs since some of them are calculated by functions and saved in the database.

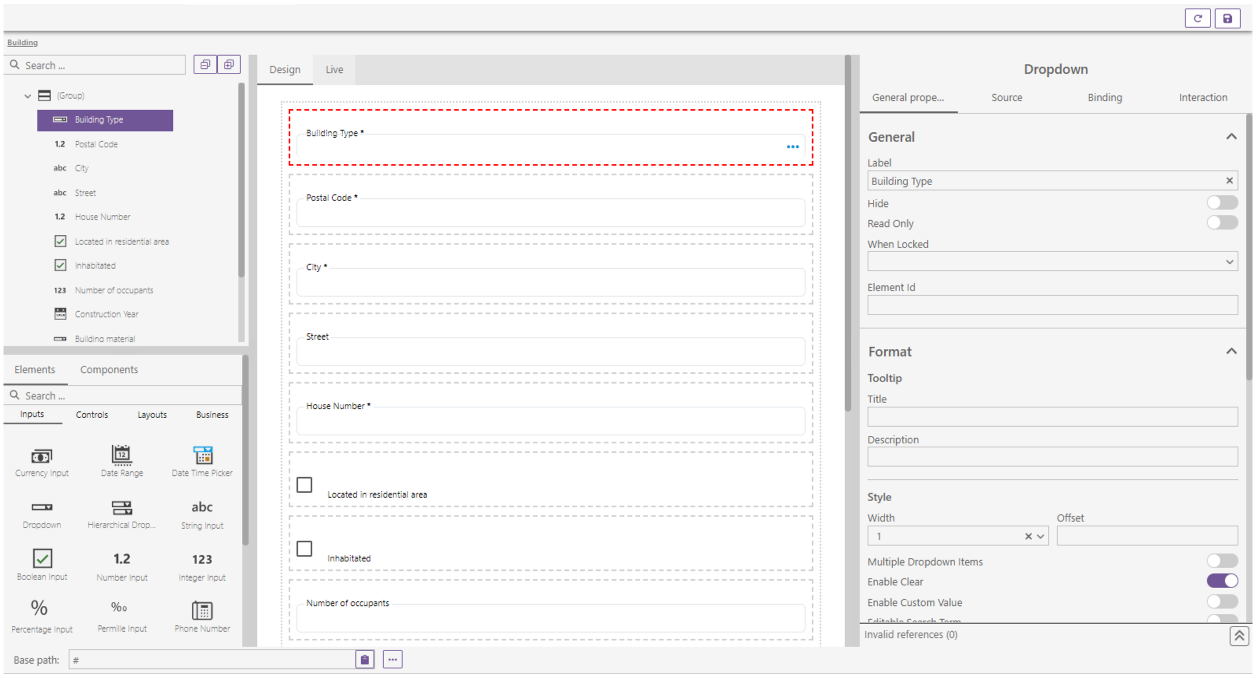
4.4.1 AdInsure

In AdInsure Studio, it is desirable to start the configuration of a new product with the UI design. The reason for this is that for every input field added to the UI is generated a property in the data model. With the help of the non-life insurance product wizard, two components were generated: the Building component with insured object type and the Coverage Options component with coverage attributes type. I started configuring the Building component, where I created all fields related to the insured building. However, the name generated in the data model is not related to the given label of the input field in the UI editor, so the user needs to change that in order to achieve business value. In the Coverage Options component, I configured a section that contained coverage options that can be added as additional coverages on the quote.

Figure 12 displays the Building component opened in the UI editor in AdInsure Studio. There are two modes to view the configured result: design and live. In the design view, the user can drag and drop elements and components directly to the modelling space displayed in the middle of the grid. Above the elements and components, it is displayed the structure of the document or component elements, and the user can easily navigate through them. For each element can be specified general properties, binding to the data model property, validation, and interactions such as events, client actions and rules. This can be specified in the right grid. After adding some inputs in the UI editor and saving, the data model is updated with properties corresponding to the added inputs in UI editor. The live mode shows how the UI will look like to the end-user if the application is published. Not all required fields added on the UI are used for premium calculation, for example, street name, house number, postal code, and city.

The quote generated by the non-life sales product wizard had a predefined UI built from standard and system components. The quote contained five tabs: participants, terms, insure objects, payment terms and ownership. Building and Coverage Options components, that were generated by the wizard, were referenced in the insured objects tab. However, I needed to change the payment terms component that grouped input fields such as payer, payment currency, payment frequency, payment until, payment type, first instalment due date and number of instalments. In the business scenario, I didn’t need to display some of the fields, such as the number of instalments and payment until date. Additionally, the payment frequency values were different from the ones provided by the component present in AdInsure. Since the component was placed in the standard layer and couldn’t be changed by the user, I overridden the component in the created layer by right-clicking on the component name and choosing the override option. After that, a component with the same name appeared in the selected layer, in which I deleted the unwanted input fields in the UI editor.

Figure 12: Building component in the UI Editor in AdInsure Studio



Base path: #

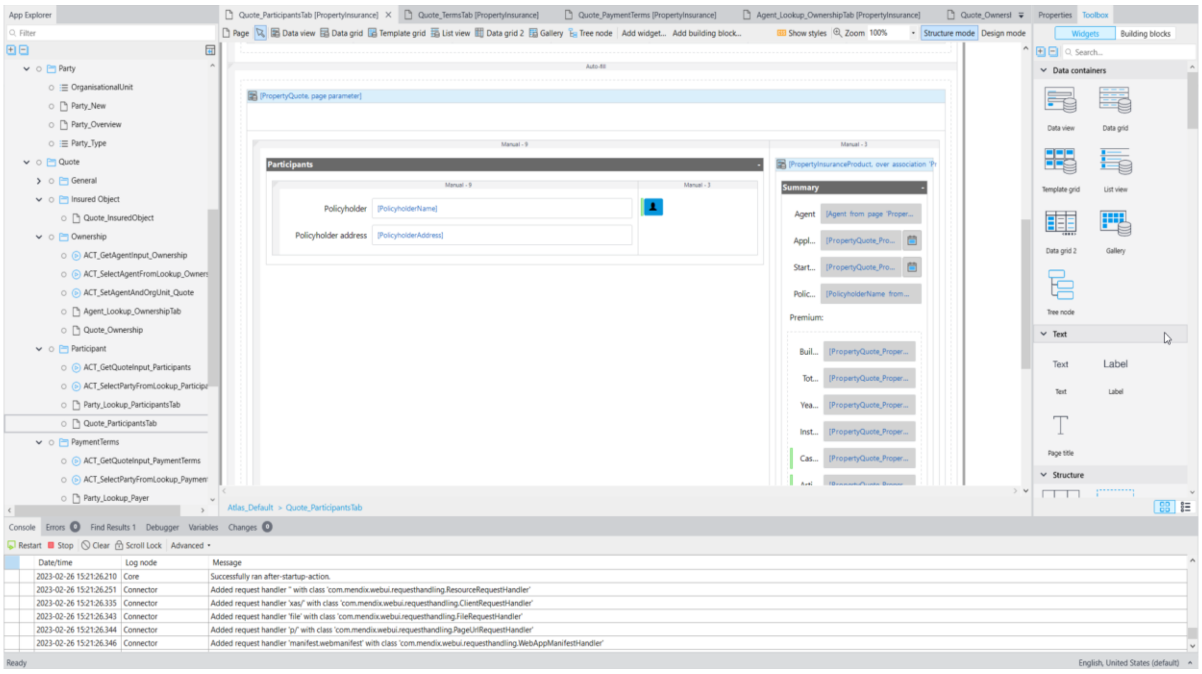
Label: Building Type
Hide
Read Only
When Locked
Element Id
Width: 1
Offset
Multiple Dropdown Items
Enable Clear ✓
Enable Custom Value

Source: Own work.

4.4.2 Mendix

Figure 13 displays the UI modeller in Mendix Studio Pro, where it is opened the Quote_ParticipantsTab page. There are also two modes that the user can switch between while modelling the page: structure and design mode. In structure mode, the user can configure and see the elements used on the page and their relations. As displayed in Figure 13, the page is opened in design mode, and it is visible that the attributes are used from the Home Quote entity (PolicyholderName and PolicyholderAddress). On the right side, it is displayed the summary section that has attributes from Property Insurance Product entity available over the association HomeQuote-to-HomeInsuranceProduct. The Design mode has the same function as the live mode in AdInsure Studio, which is a preview of the design of the page as it will look like in runtime. As mentioned in Section 3.2.2, one of the components in the Mendix Platform is Atlas UI which offers a variety of templates, layouts, widgets and building blocks. Moreover, if the offered assets are not fulfilling the application requirements, the user can create new templates and building blocks that can be reused. The widgets and building blocks are displayed in the right panel in Figure 13 and can be dragged and dropped to the design area. I used a header building block for each page where are grouped a header and a drop-down filter. The editing of the fields, such as setting labels, default values, adding events, visibility conditions and so on, can be handled in the right panel by switching to the properties tab or in a pop-up window by double-clicking on the property, which I found very useful. In the left panel in Figure 13 is visible the structure that I created for the Home Quote. I created a page for each tab, and I grouped all files related to a page in a folder.

Figure 13: Participants Tab in UI modeller in the Mendix Studio Pro



```
Date/time                Log node    Message
2023-02-26 15:21:26.210  Core        Successfully ran after-startup-action.
2023-02-26 15:21:26.251  Connector   Added request handler '' with class 'com.mendix.webui.requesthandling.ResourceRequestHandler'
2023-02-26 15:21:26.335  Connector   Added request handler 'xas/' with class 'com.mendix.webui.requesthandling.ClientRequestHandler'
2023-02-26 15:21:26.343  Connector   Added request handler 'file' with class 'com.mendix.webui.requesthandling.FileRequestHandler'
2023-02-26 15:21:26.344  Connector   Added request handler 'p/' with class 'com.mendix.webui.requesthandling.PageUrlRequestHandler'
2023-02-26 15:21:26.346  Connector   Added request handler 'manifest.webmanifest' with class 'com.mendix.webui.requesthandling.WebAppManifestHandler'
```

Source: Own work.

Besides the pages that represent the quote, I also designed the menu, dashboard, overviews, and pop-up windows used for look-up buttons. In the menu, I added five sections where I grouped all pages: dashboard, accounts, party, contracts, and agent. On the dashboard, I displayed the options for the creation of a new home quote, agent, party, and all overviews such as agent, contract, and party overview. Each overview is related to an entity, and the user can filter the saved objects in the database. For example, the Contract overview has a data grid related to the Home Quote entity, and it has four filters: quote number, start date, state, and policyholder. Pop-up windows were used for look-up buttons, such as the policyholder button. The pop-up windows contain data grids similar to the ones used on the overview pages. The difference is that in the pop-up window, the user can select one of the results displayed in the data grid, and that result will be set as a value to an input field. For example, when the user clicks on the button next to the policyholder, a pop-up window will open with the overview of entered parties in the database. When the user selects a party, the party will be set as a policyholder on the quote.

The difference between AdInsure Studio and Mendix Studio Pro UI design is that in the former were designed insurance-specific components that can be used on documents, and in the latter were designed building blocks that are not specific to a particular industry and can be used on pages for many purposes. An example of this is the Organisation component in AdInsure that I used on the ownership tab of the Quote document. In this component are grouped input fields for agent and organisation unit. Besides the visible information about the agent on the UI, such as the agent's name and organisation unit, there is other information stored in the database, such as the agent's code.

4.5 Workflows and definition of business logic

The inputs defined on the UI are needed as inputs in the functions that calculate the main output of the business scenario: the insurance premium. Since AdInsure and Mendix are LCDPs, the user is supposed to configure the business logic in the editor with minimal usage of programming.

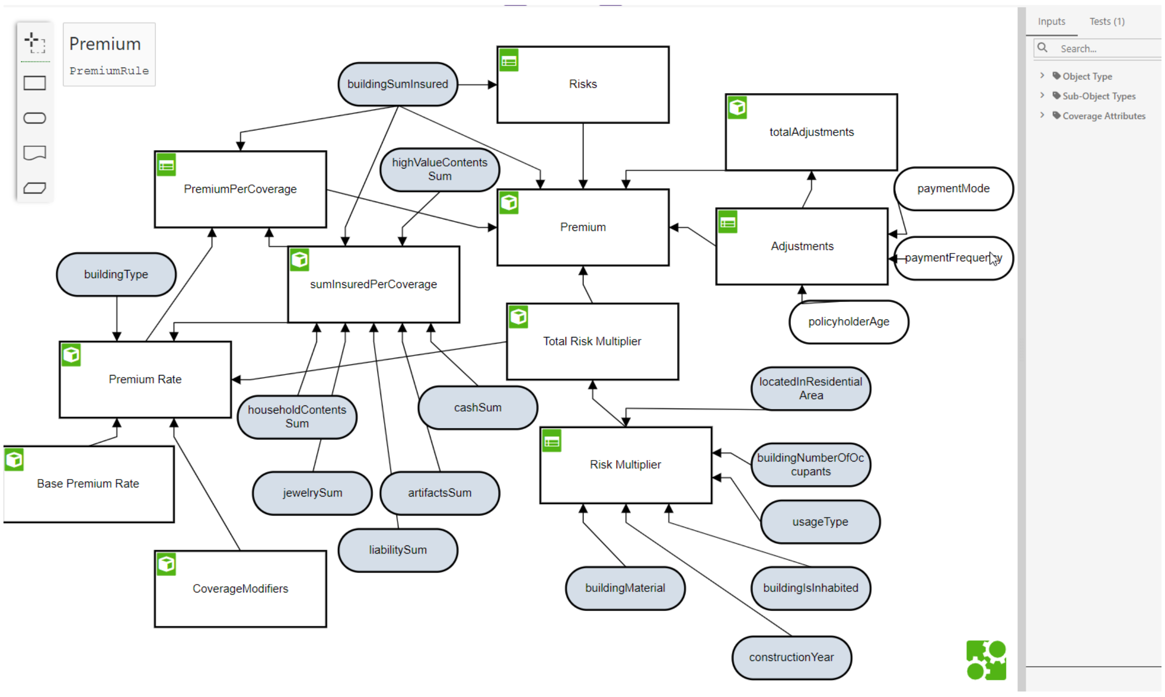
4.5.1 AdInsure

In AdInsure terminology, the insurance product is composed of many rules that are required for the creation of a functional product. Before its definition, there are specific requirements that must be completed. Firstly, all rules related to the business and technical side of the product should be defined. For example, rules related to premium duration, invoicing, currencies, taxes, underwriting, retention etc. Additionally, there should also be defined code tables that are important for every insurance product, such as risks, object types, insurance classes etc. Every insurance product is part of an insurance line, so the existence of the insurance line is also a prerequisite for the creation of an insurance product. Moreover, the insurance lines have many tariffs groups and similar insurance products are grouped under the same tariff group.

The business rules in AdInsure are configured in Decision Model and Notation (DMN), which is a modelling language and notation (OMG, n.d.). AdInsure Studio uses a DMN viewer and editor for rules preview and configuration. It is used by almost all platform elements of AdInsure (tariffs, components, views, etc.). The context of the DMN is compiled to JavaScript.

Figure 14 displays the Premium rule of the Household tariff. Each rectangle represents a decision that has one of the three types: decision table, literal expression, or context expression. The decision table type has a table icon in the left corner, and it represents a table, where each row specifies conditions and outcomes. For example, risks are represented in decision table that determines the risk code and the sum insured of the risks. Literal expression type is identified by the curly brackets icon in the left corner and is used for complex calculations. Context expression type has a cube icon and represents a set of variable names and their calculated values. Total adjustments are calculated in context expression type. The model of related decisions has a significant advantage because the user can see and understand how all elements are connected without the need to open different documents to get to know the structure of the product. The inputs are represented by rectangles with round corners. They are elements connected to the decisions and they come from Building and Coverage Options components. The rule contains all calculations for outputs required by the AdInsure platform, such as premium, tariff rate, risks, currency, etc., that are configured in the Premium decision. The user can create a decision for each element calculation that, at the end, will be used in the Premium decision. Expressions in the decisions can be written in JavaScript or in FEEL (Friendly Enough Expression Language), which is a language that is easily understood by business users.

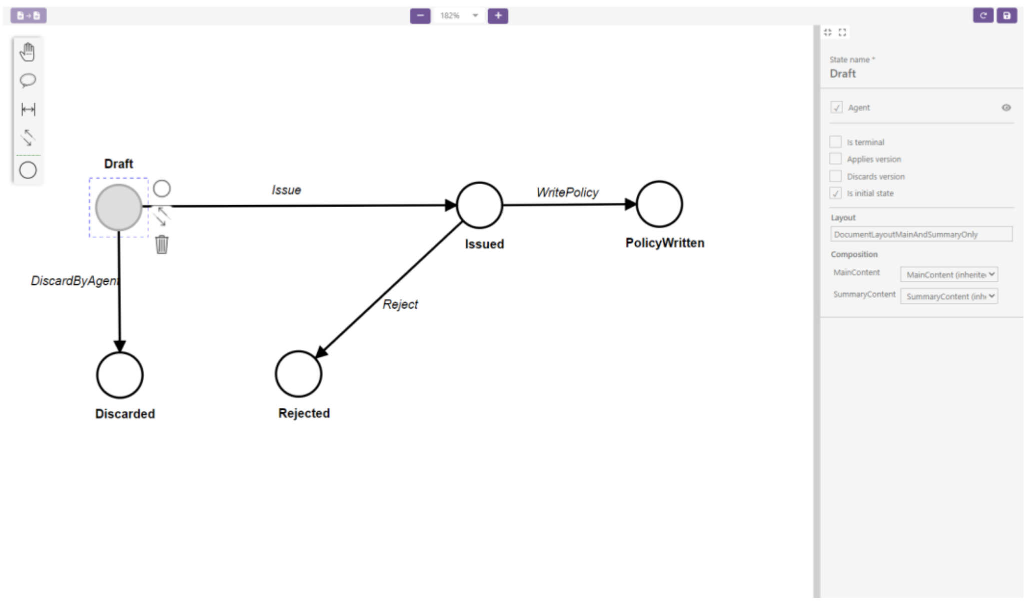
Figure 14: Premium rule of the Household tariff in AdInsure Studio



Source: Own work.

The document flow of the Home Quote is displayed in Figure 15. It is a rule applicable to sales products only. In the document flow, the user can configure the flow of a particular document and their relations with other documents. The editor contains states, transitions, and relations with other documents. Each document state must have a selected actor. The user can select many options that apply for the selected actor in a particular state, such as which transitions are available for the selected actor, available operations (such as save, calculate, print etc.), restrictions regarding the available attachments and activities in the selected state, applicable UI form and commenting availability.

Figure 15: Document flow of the Home Quote in AdInsure Studio



State name *
Draft

[✓] Agent

[ ] Is terminal
[ ] Applies version
[ ] Discards version
[✓] Is initial state

Layout
DocumentLayoutMainAndSummaryOnly

Composition
MainContent: MainContent (inherite
SummaryContent: SummaryContent (inh

Source: Own work.

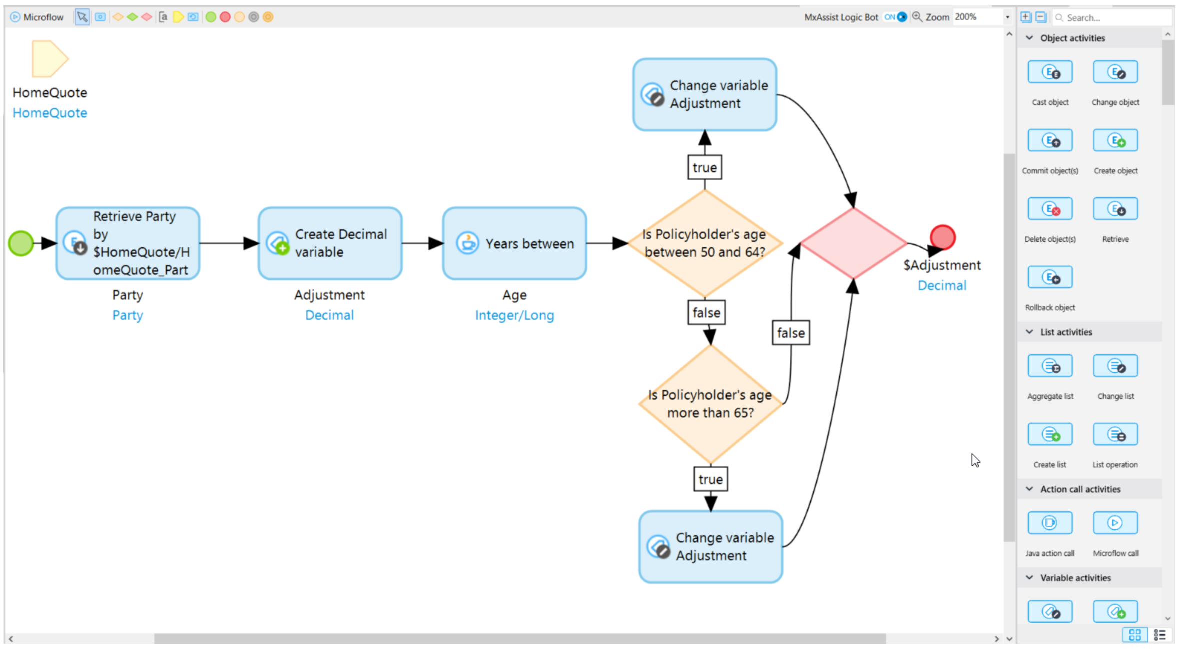
4.5.2 Mendix

In Mendix Studio Pro, the business logic is modelled through microflows, nanoflows and workflows. The first two options are related to some actions that are performed on an object or page, such as creation, deletion, update etc. The third option is related to the workflow of the application based on user roles and tasks (Mendix, 2022c). In the configuration of the business scenario, I only used microflows to define business processes as well as determine attribute value through calculation, the behaviour of UI elements such as buttons and integration with other services.

In Figure 16 is displayed the microflow that calculates the premium adjustments. The yellow parameter indicates the input in the microflow, which is the Home Quote entity. The blue rectangles are the activities that indicate some action. As displayed in the right panel in Figure 16, there are a variety of activities which the user can choose from. The figures with diamond shapes are decisions. The start of this microflow is indicated by the green dot, and the end with the red dot. The adjustment to the premium is related to the policyholder age input, which is an attribute of the Party entity, so I retrieved the Party object. The final result of this attribute is the adjustment value. This microflow is called in the main microflow, which is CAL_Premium, where is used the adjustment value for the calculation of the premium for the insured object.

In Mendix, there is a market store where the user can download various widgets and upgrade the UI and the functionality of the app. In my project, I used a module called Community Commons that contained many useful Java methods which can be used in defining the application logic. I used the YearsBetween function in order to calculate the age of the insured person, which was relevant for the adjustments value. This is the third activity in Figure 16.

Figure 16: Premium adjustments microflow in Mendix Studio Pro

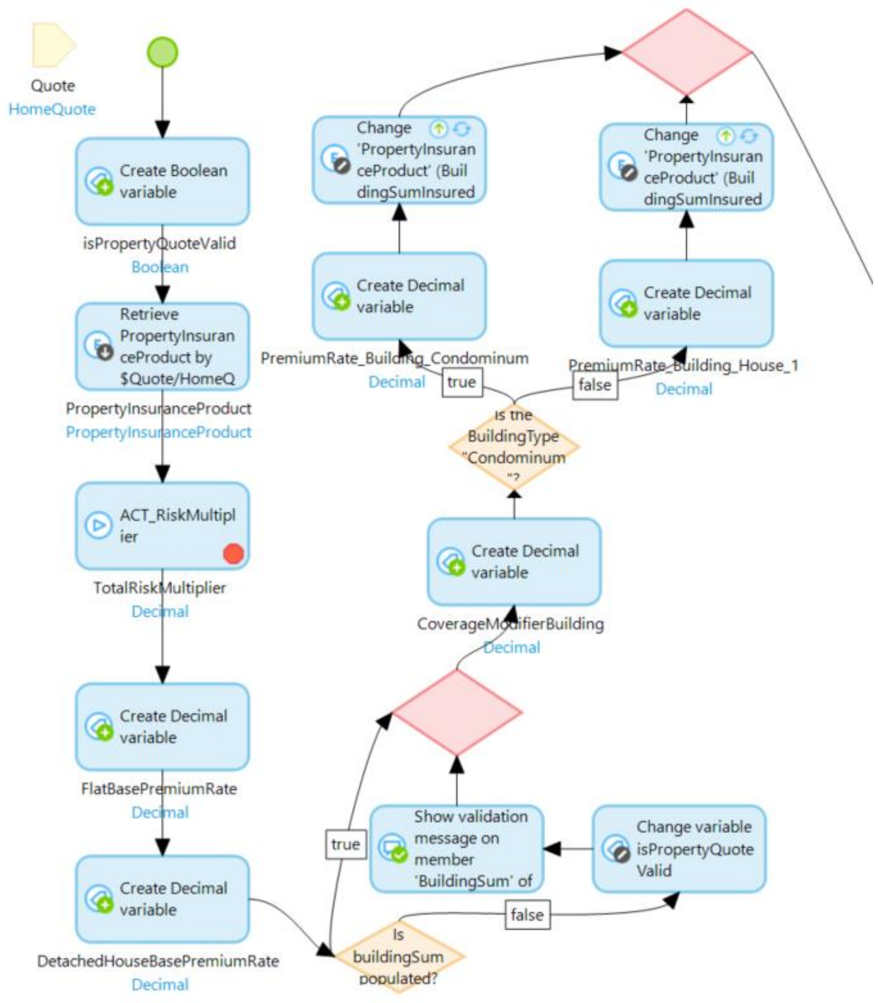


Source: Own work.

Moreover, there is an AI-Assisted development (AIAD) available through MxAssist Logic Bot, MxAssist Performance Bot and Validation Assist that helps users to develop better models. One of the biggest benefits of this is the development of models without spending too much time. In my case, I used the MxAssist Logic Bot the most. Since I didn't have any experience with Mendix, it helped me to model microflows faster by suggesting the next activity (Mendix, 2022a).

Figure 17 displays the first part of the CAL_Premium microflow, where the building sum insured is calculated. At the beginning of this microflow is called ACT_RiskMultiplier microflow, which is the third activity. The calculation of the building sum insured starts with the first decision (Is the building sum populated). The red diamond shape indicates the merge of the flows, which relation continues to another decision that checks if the sum insured of the next coverage was populated, which is the cash sum. Since there are many inputs on the quote that are used for premium calculation, I needed to create a decision for each sum insured input and then separately calculate sum insured for each coverage, which means I created six additional flows similar to the building calculation flow. By using the activity Create Decimal variable, I created a premium rate and coverage modifier variables for each coverage where I calculated their values. This resulted in a big microflow with many activities and decisions. The logic of premium calculation was not so clear as in the microflow for adjustments displayed in Figure 16 since the premium calculation was more complex than the adjustments calculation. In AdInsure, this calculation was handled through a decision table or context expression that contained all calculations related to a parameter. For example, the Premium Rate context expression in Figure 14 contained calculations of the premium rate for each coverage that was used in the Premium Per Coverage decision table.

Figure 17: First part of the Premium microflow in Mendix Studio Pro


Quote
HomeQuote
Create Boolean variable
isPropertyQuoteValid
Boolean
Retrieve PropertyInsuranceProduct by $Quote/HomeQ
PropertyInsuranceProduct
PropertyInsuranceProduct
ACT_RiskMultiplier
TotalRiskMultiplier
Decimal
Create Decimal variable
FlatBasePremiumRate
Decimal
Create Decimal variable
DetachedHouseBasePremiumRate
Decimal
Change 'PropertyInsuranceProduct' (BuildingSumInsured
Create Decimal variable
PremiumRate_Building_Condominum
Decimal
true
Change 'PropertyInsuranceProduct' (BuildingSumInsured
Create Decimal variable
PremiumRate_Building_House_1
Decimal
false
Is the BuildingType "Condominum"?
Create Decimal variable
CoverageModifierBuilding
Decimal
true
Show validation message on member 'BuildingSum' of
Change variable isPropertyQuoteValid
false
Is buildingSum populated?

Source: Own work.

Despite using microflows for calculation, I also used them for setting action to lookup buttons such as policyholder, transition buttons such as action button for a transition of the quote into a new state and save button. In some activities, I needed to write a simple expression to achieve the desired result. The functions that I needed were well documented in the Mendix documentation portal. However, an understanding of basic concepts in programming is needed to write an expression in microflow. The events triggered with the click of a button in AdInsure are handled through client actions that are written only in JavaScript.

Changing a name of an element in Mendix, such as microflow or variable, automatically updates all names of the changed elements in all places. I found this very helpful and time-saving. This functionality is not present in AdInsure, so the user needs to go through all errors produced by the Studio and update all elements. Additionally, the property could be used in some mapping fiction which will consequentially produce errors, so the user needs to have programming skills and debug in order to find the error.

Another thing worth mentioning in this section that is very important and necessary for configuration and modelling is the debugging process. During the development of an application, it is very common that the application will have a bug and it won't work as described in the business requirements. To identify the bug, the user must debug it. In AdInsure, the debugging process is a disadvantage. Since the DMN files are compiled from dmn.xml to JavaScript files, only the generated JavaScript code can be debugged. In Mendix, graphical debugging is available by setting breakpoints on specific activities in the microflow.

4.6 Integration with other applications

The possibility of integration with other applications is an important functionality that LCDPs should offer. The LCPDs strive to offer options for exposing data and services to other systems. The modelling and configuration of the Household tariff and Home insurance quote did not require integration with other applications, so this functionality was not tested in practice; however, both platforms offer this possibility.

4.6.1 AdInsure

In AdInsure are used Representational State Transfer (REST) APIs. As mentioned in Section 3.1.1, AdInsure is constructed of business modules that are decoupled. The modules use APIs, called Shared APIs in the AdInsure dictionary, to communicate with each other. Moreover, the modules can integrate with another system. This means that there is also a possibility for companies to integrate and use only one of the modules and not use the whole package, for example, the Claims module (Adacta, 2022d).

4.6.2 Mendix

REST APIs, Simple Object Access Protocol (SOAP) web services, and OData are the tools provided by Mendix for integration. Moreover, the user can choose and download connectors from the Marketplace. Additionally, there is also a possibility of developing new connectors customised to specific requirements. Integration with other systems is a complex activity, and this task involves experienced programmers. However, Mendix has widened the circle of potential task carriers by including inexperienced programmers since it includes the modelling principle even in this case (Mendix, n.d. -c).

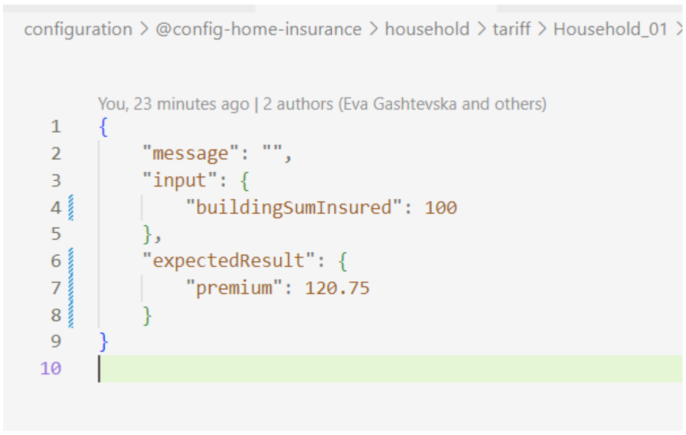
4.7 Testing

The business scenario that I implemented outputs a calculation of premium per coverage that is subject to several inputs. If there is an issue with some UI elements or rules, this will be easily spotted in the runtime by displaying an error message. However, if the premium calculation is incorrect because of changes in some variables that influence the premium, it will not be so obvious. Because of that, it is necessary to implement test scenarios that will test the premium calculation in order to indicate bugs in the early stages.

4.7.1 AdInsure

In AdInsure, this is handled through test scenarios in JSON format where the user sets an example of inputs and adds what is the expected result if these inputs are used for calculation. With the generation of tariff through the non-life insurance wizard, an empty test scenario file is generated. After tariff configuration, the user should also configure the test scenario to ensure that the tariff is correctly configured. Figure 18 displays a test scenario that tests the premium result in case the building sum insured is 100.

Figure 18: Test scenario in AdInsure Studio



configuration > @config-home-insurance > household > tariff > Household_01 >

You, 23 minutes ago | 2 authors (Eva Gashtevska and others)

```json
{
    "message": "",
    "input": {
        "buildingSumInsured": 100
    },
    "expectedResult": {
        "premium": 120.75
    }
}

```

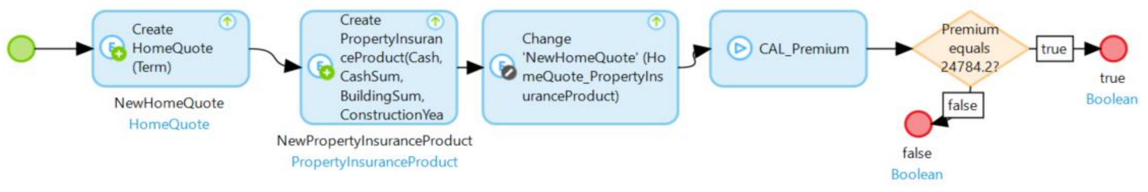
Source: Own work.

Testing of sales products is also possible through test scenarios that are different from those used for testing tariffs. Since the quotes have many inputs and outputs, they are split into two files: an example file for inputs and a scenario for outputs, which is referenced in an example file. The user must know the platform elements and how the evaluation of the insured objects is handled by the platform in order to test this result. This type of test is called validation test and it tests the configuration part. There are also API, UI, and performance tests.

4.7.2 Mendix

In Mendix there are many tools available to the user that can be used for testing the application. One of them is unit test offered as a module that can be downloaded from Mendix Marketplace and can be used to test microflows. Figure 19 displays a test microflow called Test_PremiumAmount that I created by using the Unit Testing module. The microflow tests the amount of total premium for a combination of inputs. The User Testing module also contains an overview that lists created unit tests, which I placed on the dashboard for the administrator user. The tests can be run from the dashboard, and the details of the test results can be accessed there.

Figure 19: Test for premium amount in Mendix Studio Pro


Create HomeQuote (Term)
NewHomeQuote
HomeQuote
Create PropertyInsuranceProduct(Cash, CashSum, BuildingSum, ConstructionYea
NewPropertyInsuranceProduct
PropertyInsuranceProduct
Change 'NewHomeQuote' (HomeQuote_PropertyInsuranceProduct)
CAL_Premium
Premium equals 24784.2?
true
false
true
Boolean
false
Boolean

Source: Own work.

Mendix also offers add-ons used for testing, such as Application Test Suite (ATS), which is used for automated testing. Since this add-on couldn't be used for free, it was not tested.

4.8 Security

The process of concluding an insurance contract could include many actors. In the business requirements, there are two main actors for the Home Quote: agent and customer. They can see different things in the quote and can take different actions. This is part of the application security where the user must configure the permissions for each application role. Despite the customer and agent roles, there is an administrator role that can perform all actions.

4.8.1 AdInsure

Access to configuration elements in AdInsure is managed through permissions that can be assigned to application roles. Permissions can be configured in the authorisations CSV files in each folder where there is a concept, such as a document or a view. For each application role added in the authorisations file must be specified an actor. I added the Sales Person application role to have permission for Home Quote and assigned the Agent actor to it. There are also application user groups that combine a set of application roles which can be assigned to specific users. The actions and operations available to a specific document are specified in the document flow. Hiding or showing of a specific part of the UI for some actors can be managed through Client actions that are written in JavaScript. Moreover, if UI for the actors has a lot of differences, it is possible to create a different UI schema for each actor. I used one UI schema for both actors since the same UI elements should be displayed for the agent and the customer.

4.8.2 Mendix

In Mendix, there are user and module roles. The module role is related to the access rights for a specific module, and it can be configured in a security pop-up window that is positioned in the module artifacts list and is mandatory for each module. A user role is configured on the project level, and it can have many module roles. For example, I configured the customer user to have three module roles: System.User, PropertyInsurance.Customer and CommunityCommoins.Customer. The system module is included by default in Mendix, and it has a user module role that performs normal actions in the application. The customer user in the Property Insurance module can access specific pages, microflows, and it has limited access for reading or writing an entity’s attributes. For example, customers can sign the quote, but they cannot issue or reject the quote. This action can be executed only by the agent. The Community Commons module contains functions that are used in microflows triggered by the customer, so because of that, I also added this module role to the customer user (Mendix, 2023b).

5 MAIN FINDINGS

While developing the Home insurance product and quote with Mendix Studio Pro and AdInsure Studio, I noticed many similarities and differences between the tools. In the following sections are presented three tables for each business requirement group with key similarities and differences between Mendix and AdInsure in each development stage. There are two development stages that are excluded from the tables: integration with other applications and deployment. The reason for its exclusion is that they are general stages that cannot be analysed on the business requirement level. As described in section 4.6, integration with other applications was not tested in practice. Regarding the deployment stage, both applications were deployed on-premises, but they also support other deployment options. In the development stages, I also added two additional points for comparison: security and quality assurance. Security is added to each table since I configured security for each group of business requirements. Quality assurance was tested only on the insurance product level, and because of that it is placed only in that table.

Before starting with the first development stage, which is data modelling, I generated sales and insurance products by using the wizard in AdInsure. This was a big plus since I didn’t need to create everything from the ground up. The insurance company can create new sales and insurance products very easily and fast as long as these products are not complicated. If the standard or system layer configuration does not cover the business requirements, the configurators must create new ones, which cannot be done quickly. AdInsure is a complex system that requires knowledge of insurance. Besides the intuitive editors and fast configuration of products and tariffs, the user must know what components, libraries or data sources are available in standard and system layer and for what purpose are used in order to create the optimal product. Additionally, the knowledge base and documentation that supports the user are not developed to the level where the configurator can work without any help from developers that are experienced in using this platform. This means the learning curve for using this platform is not so fast. Additionally, it is important to mention that the configuration process was faster in Mendix than in AdInsure Studio.

Mendix, on the other hand, has a very structured and easy-to-understand documentation and knowledge base. A feature that I found very useful is the help icon on every pop-up window in Medix that is related to setting or editing a particular element, that leads to the official documentation site. Additionally, the Mendix community is big, and the user can search for particular term in Mendix Academy, documentation, forum, or ask a direct question in the forum. During modelling in Mendix, all questions and issues that I encountered were explained and answered in the documentation or in the Mendix Forum.

In Table 2 are displayed the key differences between Mendix and AdInsure in each development stage of the sales product.

Table 2: Differences in the development of the sales product between Mendix and AdInsure

Development stage	**Mendix**	**AdInsure**
Data modelling	– Definition of entities and their associations	– Use of pre-defined insurance-specific components with ready data models. – Available extendibility of predefined components (by overriding)
User Interface design	– Use of building blocks and widgets – Access to Mendix Marketplace for ready-to-use components	– Use of predefined components by non-life sales wizard – Use of predefined insurance-specific components – Available extendibility of components – Sales product is a document with a sequence number
Business logic definition	– Logic defined visually with Microflows. – Possibility to use Nanoflows and Workflows	– Defined in JavaScript – Document flow definition in document flow editor (BPMN)

table continues

Table 2: Differences in the development of the sales product between Mendix and AdInsure (continued)

Development stage	**Mendix**	**AdInsure**
Business logic definition	– Possibility to use JavaScript and Java – Access to Mendix Marketplace for ready-to-use components – AI-Assisted development – Sync of naming changes during development. – Debugging	
Security	– Configuration of security for each module and element – Definition of user and module roles	– Definition of security in authorization file in .csv format for each platform element. – Defined application roles and application user groups

Source: Own work.

In data modelling stage in Mendix was defined the data model with its entities and associations between them. I did not need to write any expressions or create tables since the tables are automatically generated. In AdInsure there was no need to create data models for components related to the sales product, since the sales product was generated by the wizard and all components referenced there were from standard and system layer and had defined data models. Regarding the UI of the sales products, I used building blocks and widgets to build it.

The UI modelling in both platforms is very similar. The UI modelling in Mendix is easy and intuitive. All properties related to the added widget or building block can be edited in a pop-up window with a double click on the widget or building block. Compared to the AdInsure UI editor, in Mendix, the user can set more properties than in AdInsure. Additionally, the user can choose between many UI templates. The UI editor in AdInsure has a big advantage since it could automatically generate properties in the data model. Moreover, the generated sales product is a document with a configurable numbering rule which adds additional business value.

The definition of the business logic in Mendix and AdInsure was different. In Mendix, I used microflows and simple expressions within the added activities. However, there were other ways available for logic definition, such as workflows and nanoflows, as well as JavaScript and Java functions. The modelling process was simple by using the MxAssist Logic Bot. Also, changing the names of entities, properties, or microflows did not cause any problems since the new name was applied in all elements that used the platform element. An additional feature that Mendix has is a friendly debugging process since the user can add breakpoints on any microflow activity. In AdInsure, logic regarding sales product was defined in JavaScript except for the document flow, which was defined in document flow editor. This means that definition of logic on the sales product level, such as mapping or client actions is not so configurator friendly since they require knowledge of programming. An exception is the configuration of events and document flow.

Regarding security, both platforms offer an easy way to handle it. In Mendix, this is done by the configuration of module and user roles. The security in AdInsure is handled through the definition of the application, application user roles and configuration of authorisation files.

Table 3: Differences in the development of the insurance product between Mendix and AdInsure

Development stage	**Mendix**	**AdInsure**
Data modelling	– Definition of entities and their associations	– Generated empty data model by non-life sales wizard
User Interface design	– Use of building blocks and widgets – Access to Mendix Marketplace for ready-to-use components	– Generated empty UI schema by non-life insurance product wizard – Used elements in the UI editor for configuration
Business logic definition	– Defined visually with Microflows – Possibility to write simple expressions in microflows – Possibility to use Nanoflows and Workflows – Possibility to use JavaScript and Java – Used Marketplace module function. – AI-Assisted development – Debugging	– Defined through DMN rules – Possibility to use JavaScript or define expressions in FEEL
Security	– Configuration of security for each module and element – Definition of user and module roles	– Definition of security in authorization file in .csv format for each platform element. – Defined application roles and application user groups
Quality Assurance	– Used UnitTests module from Mendix Marketplace	– Definition of simple test scenarios in JSON format.

Source: Own work.

Table 3 displays the key differences in each platform related to configuration of the insurance product, grouped by development stage.

Data modelling, UI, business logic and security stages in Mendix for insurance product were the same as in the sales product. In AdInsure, in the data modelling stage, there were no ready-to-use data models. Instead, the wizard generated an empty data model that had to be defined.

Business logic for the insurance in AdInsure was defined through DMN rules, where the user can add business logic by using JavaScript or FEEL expressions. Compared with Microflows, the logic for the insurance product defined in DMN is clearer and easier to understand. Microflows are a great way for the definition of business logic; however, in the case of the definition of an insurance product that depends on many variables and has many combinations, AdInsure performed better. It is also important to consider that AdInsure accepts only a particular structure of insurance products and documents defined by the platform, so the insurance companies must follow the guidelines.

The UI of the insurance product covered the configuration of the UI for coverage options. The UI configuration was similar in both platforms since it included drag-and-drop method for elements to the empty pages. In Mendix, the modelling was done directly on the insurance page and in AdInsure, the UI was configured on the component level.

In the development process of the insurance product, quality assurance was added since I added tests that checked the calculation of the insurance product. In Mendix, that was achieved by using a module available on Mendix Marketplace. In AdInsure, an empty file was autogenerated by the wizard for test scenario definition. The tests in AdInsure were tariff focused and easier to define. However, in both platforms are also available other types of tests.

Table 4 defines the insured object's development stages and key differences. The insured object in AdInsure was created as a component. We can see from Table 4 that for configuration and modelling of the insured object in Mendix are listed the same features for each development stage as for the sales product. Compared to the sales product development in AdInsure, there are differences in data modelling and UI stages since the wizard generated empty files that needed to be defined. Additionally, the process of development started with the definition of the UI which automatically generated properties in the data model. In the other stages were used the same features as in the sales product.

Table 4: Differences in the development of the insured object between Mendix and AdInsure

Development stage	**Mendix**	**AdInsure**
Data modelling	– Definition of entities and their associations	– Generated empty data model by non-life insurance product wizard. – Data properties automatically generated in the UI editor

table continues

Table 4: Differences in the development of the insured object between Mendix and AdInsure (continued)

Development stage	**Mendix**	**AdInsure**
User Interface design	– Use of building blocks and widgets	– Generated empty UI schema by non-life insurance product wizard – Used elements in UI editor for configuration
Business logic definition	– Logic defined visually with Microflows – Possibility to use Nanoflows and Workflows – Possibility to use JavaScript and Java – Access to Mendix Marketplace for ready-to-use components – AI-Assisted development – Sync of naming changes during development – Debugging	Defined in JavaScript
Security	– Configuration of security for each module and element – Definition of user and module roles	– Defined in authorisation file in .csv format for each platform element. – Defined application roles and application user groups.

Source: Own work.

In AdInsure, the business users that have the configurator role are people with knowledge in the insurance field. Their role as configurator is limited since they could create or change tariffs and simple UI designs of products. For other things, such as setting events or mapping of some attributes, it is required programming knowledge. In contrast with that, Mendix has a tool for each user group: Mendix Studio and Mendix Studio Pro. This has an advantage since the users have an environment dedicated and adapted to their knowledge level. This means that business users with no technical knowledge are not exposed to advanced functionality that they do not need and cannot use. In AdInsure, business users have access to all functionalities, including the code, regardless of their programming knowledge, which is unnecessary.

6 DISCUSSION

This research has shown that an insurance product can be successfully developed in general and insurance LCDPs. The results from the comparison between the platforms' functionalities in each development cycle confirmed reaching this research's main goal, which was analysing, comparing and finding the main differences between the general and industry-based LCDPs. The insurance LCDPs offer out-of-the-box functionalities with insurance context and better editors for business rules. In contrast, general LCDPs offer more powerful UI editors, a strong community and a variety of basic functionalities that can be used to configure insurance solutions.

Despite the difference in functionalities, the development of Home insurance product in two different LCDPs confirmed their main benefits: faster development and lower costs. The solution was developed for three users in each platform: administrator, agent, and customer. The product aimed to give customers an informative premium calculation that includes the chosen coverages. However, creating and issuing a quote is only one piece of the puzzle. Insurance companies have accounting, billing, claims, policy management, sales, and reinsurance processes. Not all these processes are created and digitalised in one platform. AdInsure, as an insurance platform, offers and supports all these processes separated into modules. The new configuration produced by the AdInsure Studio is structured in a way compatible with all insurance modules. As previously said, the insurance company could only need one module to implement in their system and not all modules. In Mendix, the user could develop an application for any industry. Their focus is not on a specific industry; they have focused on offering a wider range of functionalities to satisfy the modelling of applications in each industry. Insurance companies have complex modules, and general LCDPs are not always the right fit for building a whole core system.

The appearance of new technologies on the market and their influence on the insurance industry created a space for success of innovative insurance companies. The requirements for building modern and innovative solutions cross the limits of the insurance LCDPs and create a need for integration with other platforms that offer building applications by incorporating new technologies. A typical example is the previously mentioned FaceQuote application that uses AI and ML. Since the insurance LCDPs support insurance core systems and their low-code tools support producing configuration within the platform-specific insurance context, the best way of expanding its ability is integration with general LCDPs such as Mendix. In this way, the insurance companies could build innovative solutions with not spending too much time and resources on development. In my opinion, insurance companies should consider the power of combining insurance and general LCDPs. General LCDPs are more powerful for building applications with modern UI that are suitable on any device that targets first contact with potential customers. Combined with APIs from insurance LCDPs, these applications could be built fast, giving great value to the company. It is essential to mention that all general and insurance LCDPs do not have the same spectrum of features. Even the LCDPs placed in the same group, such as industry-based or general

LCDPs, differentiate. Choosing a general and insurance LCDP is a significant investment for the company, and because of that, a previous market analysis is required. Companies should focus on finding the best combination of platforms to help them build modern and robust solutions that satisfy their stakeholders' needs.

There are many limitations related to the research. Comparison of the LCDPs' functionalities was based on the developed Home insurance product. This means that this research covered only the Sales module and the first stage of the insurance process, which is the creation of a quote. More complex scenarios were not analysed, such as policy creation and its effects on other related modules, such as Accounting and Billing modules. Additionally, integration between both platforms was not included in this research. Future research should address the identified gaps in the current research. It should include integrating industry-based and general LCDP and measure the benefits of using multiple platforms instead of one. The challenges during integration should be identified and analysed. Moreover, inclusion of scenario that covers more business modules is needed. Information about the used time for developing business solutions in different LCDPs could also be beneficial. I didn't provide this information because I had previous experience using AdInsure Studio, and the results would not be transparent.

CONCLUSION

In my master's thesis, I compared the development process in two LCDPs by implementing a Home insurance solution in both platforms. The first platform was an enterprise LCDP called Mendix, in which I used its low-code tool Mendix Studio Pro. AdInsure Studio was the low-code tool of the second platform I used, an insurance platform called AdInsure. With the analysis, I aimed to answer the research question, which was to find the key differences between using industry-oriented and general LCDPs for developing insurance solution, based on the comparison between Mendix and AdInsure platforms. At the beginning of my master's thesis, was presented the theoretical part, which consisted of three chapters: LCDP, insurance industry and overview of the analysed LCDPs. In the first chapter, I explained some concepts closely related to LCDPs, such as abstraction and MDD and continued with the definition of LCDPs, their architecture and development process, added a comparison of LCDPs and traditional development and analysis results of market leaders on the enterprise LCDP market. The second chapter is related to the insurance industry, where I started with an introduction to insurance, general terms of insurance and digital transformation of insurance companies. In the third chapter are explained the platforms used for the analysis and brief information on the companies that developed them is provided. The practical part is contained in the fourth chapter. At the beginning of the Chapter 4, I added the business requirements for the developed Home insurance product and continued describing the required preconditions for starting the configuration of the Home insurance product. After that, I added a development analysis in each development cycle for both platforms. In the main findings part, I answered the research questions by describing comparison results by dividing them into three groups: sales product, insurance coverage and object type.

The insurance product was successfully implemented in both platforms. However, I found out that there are many differences in the development cycles of each platform. The industry-oriented platform had predefined functionalities with an insurance context ready for use for building new insurance and sales products, such as data models, UI and business logic definition of the sales product. For the insured coverage and insured object, files were generated by the non-life insurance and sales product wizards where the user had to define the UI, business logic, security, and tests. The UI definition in both platforms was similar since platforms supported drag-and-drop functionality. However, the general platform offers a better UI configuration experience for the user since it offers a variety of templates. Business logic definition for the insurance product was more straightforward in an industry-oriented platform where it was defined in DMN rules. However, the logic related to events was better defined in the general platform since it was visually defined in microflows. Additional criteria for comparison were added security and quality assurance. Security configuration was easy in both platforms; however, the user experience for the definition of security was better in the general platform. Quality assurance, which refers to the definition of the test about calculated premium, was more straightforward in the insurance platform since the test document had only two objects that could directly define the inputs and the expected results. Despite the chosen criteria for comparison, remarkable differences related to the development process were the strong community for developers' support and the available marketplace for components, modules and projects that the general platform provides to its users.

REFERENCE LIST

1. Acko. (2022, October 14). *Types of Insurance*. Retrieved November 6, 2022, from https://www.acko.com/articles/general-info/types-of-insurance/#what-is-life-insurance

2. Adacta. (2022a). *Insurance Application*. Retrieved from Internal Portal (Adacta): Unpublished

3. Adacta. (2022b). *Insurance Quote*. Retrieved from Internal Portal (Adacta): Unpublished

4. Adacta. (2022c). *Insurance Product*. Retrieved from Internal Portal (Adacta): Unpublished

5. Adacta. (2022d). *Configurable modular platform*. Retrieved from Internal Portal (Adacta): Unpublished

6. Adacta. (2022e). *About Adacta*. Retrieved December 20, 2022, from https://www.adacta-fintech.com/about-us

7. Adacta. (2022f). *Logical Architecture*. Retrieved from Internal Portal (Adacta): Unpublished

8. Adacta. (2022g). *Concepts we follow*. Retrieved from Internal Portal (Adacta): Unpublished

9. Adacta. (2022h). *Create non-life insurance product*. Retrieved from Internal Portal (Adacta): Unpublished

10. Adinsure. (2022i). *Adinsure Studio*. Retrieved from Internal Portal (Adacta): Unpublished

11. Alamin, M. A., Malakar, S., Uddin, G., Afroz, S., Haider, T., & Iqbal, A. (2021). An Empirical Study of Developer Discussions on Low-Code Software Development Challenges. *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR),* (pp. 46-57). doi:10.1109/MSR52588.2021.00018

12. Aloqaily, M., Otoum, S., Tseng, L., & Othman, J. (2020). Blockchain for Managing Heterogeneous Internet of Things: A Perspective Architecture. *IEEE Network*. doi:10.1109/MNET.001.1900103

13. Baer, D. S. (2010). Expectations for a Fourth Generation Language. *IBM Corporation*, 755-764.

14. Bentley, J. L. (1982). *Writing Efficient Programs.* Prentice Hall Ptr.

15. Beynon-Davies, P., Mackay, H., Carne, C., & Tudhope, D. (1999, September). Rapid application development (RAD): an empirical review. *European Journal of Information Systems, 8,* 211-222.

16. Bloomberg. (2020, September 15). *COVID-19 Accelerates Insurance Digitalization to Meet Customer Demand: World InsurTech Report 2020*. Retrieved December 10, 2022, from Bloomberg: https://www.bloomberg.com/press-releases/2020-09-15/covid-19-accelerates-insurance-digitalization-to-meet-customer-demand-world-insurtech-report-2020

17. Bock, A. C., & Frank, U. (2021a). In Search of the Essence of Low-Code: An Exploratory Study of Seven Development Platforms. *2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Compan*, 57-66.

18. Bock, A., & Frank, U. (2021b, January). Low-Code Platform. *Business & Information Systems Engineering*, 733-740.

19. Brown, A. W., Conallen, J., & Tropeano, D. (2005). Introduction: Models, Modeling, and Model-Driven Architecture (MDA). In S. Beydeda, M. Book, & V. Gruhn (Eds.), *Model-Driven Software Development* (pp. 1-16). Berlin, Heidelberg: Springer.

20. Cabot, J. (2020, October). Positioning of the low-code movement within the field of model-driven engineering. *ACM/IEEE 23rd International Conference on Model Driven Engineering Languages and Systems (MODELS '20 Companion)*, (pp. 1-3).

21. Carroll, N., Móráin, L., Garrett, D., & Jamnadass, A. (2021, April). The Importance of Citizen Development for Digital Transformation. *Cutter IT Journal, 34*, 5-9.

22. Catlin, T., Lorenz, J.-T., Münstermann, B., & Ricciardi, V. (2017, March 1). *Insurtech—the threat that inspires*. Retrieved December 12, 2022, from McKinsey: https://www.mckinsey.com/industries/financial-services/our-insights/insurtech-the-threat-that-inspires

23. Chen, Y., Dios, R., Mili, A., Wu, L., & Wang, K. (2005, June). An Empirical Study of Programming Language Trends. *IEEE Software, 22,* 72- 79.

24. Clay, R., & Rymerwith, J. R. (2016). The Forrester Wave™: Low-Code Development Platforms, Q2 2016.

25. Crunchbase. (n.d.). *Mendix*. Retrieved 28 6, 2023, from Crunchbase: https://www.crunchbase.com/organization/mendix

26. Daly, M. (2020, April 22). *No-Code platforms the beating heart of insurance innovation*. Retrieved July 28, 2021, from InsureTech World: https://www.insurtechworld.org/post/102g5dz/no-code-platforms-the-beating-heart-of-insurance-innovation

27. Damaševičius, R. (2006, January). On the quantitative estimation of abstraction level increase in metaprograms. *Computer Science and Information Systems, 3*, 53-64. doi:10.2298/CSIS0601053D

28. Di Ruscio, D., Kolovos, D., Lara, J., Pierantonio, A., Tisi, M., & Wimmer, M. (2022). Low-code development and model-driven engineering: Two sides ofthe same coin? *Software and Systems Modeling, 21,* 437–446.

29. Dorfman, M. S. (1998). *Introduction to Risk Management and Insurance* (6th ed.). Prentice Hall, Inc.

30. EIOPA. (2022, September 20). *European Insurance Overview 2022*. Retrieved November 6, 2022, from https://www.eiopa.europa.eu/document-library/report/european-insurance-overview-2022

31. ERP-One. (2020, August 14). *The rise of no-code*. Retrieved February 20, 2022, from ERP-One: https://erp-one.com/the-rise-of-no-code/

32. Fabozzi, F. J., & Drake, P. P. (2010). Domestic Financial Sector. In *The Basics of Finance: An Introduction to Financial Markets, Business Finance, and Portfolio Management 1st Edition* (Vol. 1, pp. 43-60). Hoboken, New Jersey, United States of America: John Wiley & Sons.

33. Feyen, E., Lester, R., & Rocha, R. (2011, February). What Drives the Development of the Insurance Sector? An Empirical Analysis Based on a Panel of Developed and Developing Countries. *Journal of Financial Perspectives, 1*, 1-43.

34. Gartner. (2020, September 30). Magic Quadrant for Enterprise Low-Code Application Platforms. Retrieved March 3, 2021, from https://www.gartner.com/doc/3991199

35. Goldberg, J. (2021, September 27). *Are Low-Code/No-Code Platforms the New Visual Basic?* Retrieved December 4, 2022, from AiteNovarica Group: https://aite-novarica.com/blogs/jeff-goldberg/are-low-codeno-code-platforms-new-visual-basic

36. Google Trends. (n.d.). Low-code development platforms. Retrieved October 22, 2022, from https://trends.google.com/trends/explore?date=2016-01-01%202022-01-10&q=%2Fg%2F11c6cx4nrr&hl=en-US

37. Grace, M., & Klein, R. W. (2003, August 28). Homeowners Insurance: Market Trends, Issues and Problems. *SSRN Electronic Journal*, 1-98. doi:http://dx.doi.org/10.2139/ssrn.816927

38. Gupta, P. K. (2008). *Fundamentals of Insurance.* New Delhi, India: Himalaya Publishing House.

39. Hailpern, B., & Tarr, P. (2006, February). Model-driven development: The good, the bad, and the ugly. *IBM Systems Journal, 45*(3), 451-461.

40. Heller, M. (2021, November 10). *Microsoft Power Apps review: Sweeter than Honeycode*. Retrieved October 16, 2022, from InfoWorld: https://www.infoworld.com/article/3638115/microsoft-power-apps-review-sweeter-than-honeycode.html

41. Kačar, T. (2010). *Osnove zavarovalništva* (1st ed.). Ljubljana, Slovenia: Slovensko zavarovalno združenje.

42. Kahanwal, B. (2013, October). Abstraction Level Taxonomy of Programming Language Frameworks. *International Journal of Programming Languages and Applications (IJPLA), 3*.

43. Kardoš, M., & Drozdova, M. (2010, June). Analytical method of CIM to PIM transformation in Model Driven Architecture (MDA). *Journal of Information and Organizational Sciences, 34*, 88-99.

44. Luo, Y., Liang, P., Wang, C., Shahin, M., & Zhan, J. (2021). Characteristics and Challenges of Low-Code Development: The Practitioners' Perspective. *15th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM).* doi:10.1145/3475716.3475782

45. Maier, P., Ulrich, F., & Bock, A. (2021). *Low code platforms: Promises, concepts and prospects. A comparative study of ten systems.* Essen: Universität Duisburg-Essen, Institut für Informatik und Wirtschaftsinformatik (ICB).

46. McKinsey. (2015). *Insurance on the threshold of digitization: Implications for the Life and P&C workforce.* Retrieved December 12, 2022, from https://www.mckinsey.com/~/media/mckinsey/dotcom/client_service/financial%20services/latest%20thinking/insurance/insurance_on_the_threshold_of_digitization.ashx

47. McKinsey. (2017). *Digital disruption in insurance: Cutting throug the noise.* McKinsey. Retrieved March 12, 2022, from https://www.mckinsey.com/~/media/mckinsey/industries/financial%20services/our%20insights/time%20for%20insurance%20companies%20to%20face%20digital%20reality/digital-disruption-in-insurance.ashx

48. McKinsey. (2021, March 12). *Insurance 2030—The impact of AI on the future of insurance*. Retrieved November 28, 2022, from McKinsey: https://www.mckinsey.com/industries/financial-services/our-insights/insurance-2030-the-impact-of-ai-on-the-future-of-insurance

49. McLaughlin, C. (2020, December 18). *The role of low code development tools in insurance*. Retrieved December 12, 2022, from InsurTech: https://insurtechdigital.com/technology-and-ai/role-low-code-development-tools-insurance

50. Mendix. (2022a, November 17). *Mendix Assist*. Retrieved March 4, 2023, from https://www.mendix.com/evaluation-guide/app-lifecycle/test-automation-quality-assurance/

51. Mendix. (2022b, September 16). *Domain Model*. Retrieved October 22, 2022, from https://docs.mendix.com/refguide9/domain-model/#1-introduction

52. Mendix. (2022c, September 28). *Application Logic*. Retrieved Ocober 22, 2022, from https://docs.mendix.com/refguide9/application-logic/

53. Mendix. (2023a, February 20). *Developer Portal Guide*. Retrieved February 20, 2023, from Mendix: https://docs.mendix.com/developerportal/

54. Mendix. (2023b, April 20). *Security*. Retrieved April 30, 2023, from https://docs.mendix.com/refguide9/security/

55. Mendix. (2023c, February 1). *App Roles*. Retrieved May 15, 2023, from https://docs.mendix.com/developerportal/collaborate/app-roles/

56. Mendix. (n.d. -a). *New Mendix CEO Tim Srock Sets Strategic Direction for Next Phase of Hypergrowth*. Retrieved March 20, 2023, from Mendix: https://www.mendix.com/press/new-mendix-ceo-tim-srock-sets-strategic-direction-for-next-phase-of-hypergrowth/

57. Mendix. (n.d. -b). *Why Was Mendix Founded?* Retrieved January 20, 2023, from Mendix: https://www.mendix.com/evaluation-guide/why-founded/

58. Mendix. (n.d. -c). *Integration*. Retrieved February 27, 2023, from Integration: https://www.mendix.com/evaluation-guide/app-capabilities/integration/

59. Mendix. (n.d. -d). *Contact us*. Retrieved October 16, 2022, from Mendix: https://www.mendix.com/contact-us/

60. Mendix. (n.d. -e). *Become a Rapid Developer - Universities.* Retrieved October 16, 2022, from https://academy.mendix.com/link/modules/550/lectures/4291/1.3-The-Mendix-Platform

61. Mendix. (n.d. -f). *The Mendix Atlas UI Framework*. Retrieved May 15, 2023, from Mendix: https://www.mendix.com/atlas/

62. Mendix. (n.d. -g). *Citizen Development*. Retrieved October 16, 2022, from https://www.mendix.com/citizen-developers/

63. Mendix. (n.d. -g). *Getting Help with Your App*. Retrieved May 15, 2023, from https://academy.mendix.com/link/modules/550/lectures/4293/1.4-Getting-Help-with-Your-App

64. Mendix. (n.d. -h). *Zurich FaceQuote Goes from Idea to App in Weeks*. Retrieved May 15, 2023, from Mendix: https://www.mendix.com/customer-stories/zurich-facequote-goes-idea-app-weeks/

65. Mendix. (n.d. -i). *App Development*. Retrieved January 8, 2023, from https://www.mendix.com/evaluation-guide/app-lifecycle/app-development/#:~:text=Users%20of%20Mendix%20Studio%20Pro,branch%20lines%2C%20and%20manage%20security.

66. Mendix. (n.d. -i). *Developer Portal Guide*. Retrieved May 15, 2023, from https://docs.mendix.com/developerportal/

67. Mendix. (n.d. -p). *The leading enterprise*. Retrieved October 16, 2022, from Mendix: https://www.mendix.com/platform/

68. Mishra, M. N., & Mishra, S. B. (2016). Evolution of insurance. In *Insurance Principles and Practice* (22nd ed., pp. 8-15). New Delhi: S Chand Publishing.

69. Mousami. (2014). Model Driven Software Engineering. *International journal of engineering research and technology(IJERT) ETRASCT, 2*.

70. OECD. (2017). *Technology and innovation in the insurance sector*. Retrieved December 12, 2022, from https://www.oecd.org/pensions/Technology-and-innovation-in-the-insurance-sector.pdf

71. OMG. (2014, June 1). *Model Driven Architecture (MDA)*. Retrieved August 23, 2022, from MDA Guide rev. 2.0: https://www.omg.org/cgi-bin/doc?ormsc/14-06-01

72. OMG. (n.d.). *Precise specification of business decisions and business rules*. Retrieved February 26, 2023, from OMG: https://www.omg.org/dmn/

73. Outreville, J. F. (1998). Insurance Concepts. In *Theory and Practice of Insurance* (1 ed., pp. 131-146). New York: Springer. doi:10.1007/978-1-4615-6187-3

74. OutSystems. (n.d.). *Develop more apps in less time with ease*. Retrieved October 16, 2022, from OutSystems: https://www.outsystems.com/low-code-platform/accelerated-development/

75. Paetsch, F., Eberlein, A., & Maurer, F. (2007). Requirements engineering and agile software development. *Proceedings of the IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, (pp. 308 - 313). doi:10.1109/ENABL.2003.1231428

76. Pratt, M. K. (2021, March). *Low-code and no-code development platforms*. Retrieved October 22, 2022, from TechTarget: https://www.techtarget.com/searchsoftwarequality/definition/low-code-no-code-development-platform

77. PwC. (n.d.). *InsurTech's moment: Legacy companies, startups and the drive for faster, cheaper and better results*. Retrieved December 11, 2022, from PwC: https://www.pwc.com/us/en/industries/financial-services/library/insurtech-innovation.html

78. Rejda, G. E., McNamara, M. J., & Rabel, W. H. (2021). *Principles of Risk Management and Insurance* (14th Global Editon ed.). Pearson.

79. Sahay, A., Indamutsa, A., Di Ruscio, D., & Pierantonio, A. (2020, August). Supporting the understanding and comparison of low-code development platforms. *46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)* (pp. 171-178). IEEE. doi:10.1109/SEAA51224.2020.00036

80. Sahoo, S. C., & Das, S. C. (2009). *Insurance Management: Texts and Cases.* Mumbai, India: Himalaya Publishing House Pvt. Ltd.

81. Salesforce. (2021, June 30). *Accelerating Digital Transformation with Rapid App Development in Low-Code Environments*. Retrieved October 16, 2022, from Salesforce: https://www.salesforce.com/ap/blog/2021/06/accelerating-digital-transformation-with-rapid-app-development-in-low-code-environments.html

82. Sego, D. (2021, August). *Why I'm Sticking with Salesforce: A Developer's Perspective on the Low-Code Platform*. Retrieved October 16, 2022, from Salesforce: https://www.salesforce.com/eu/blog/2021/08/developer-perspective-low-code-platform.html

83. Selic, B. (2003). The pragmatics of model-driven development. *IEEE Software, 20*, 19-25.

84. Shakeel, F. (2022, October 17). *Low-code in Insurance: Achieving Accelerated Modernization in 2023*. Retrieved December 12, 2022, from Damco: https://www.damcogroup.com/blogs/accelerate-modernization-with-low-code-insurance-platforms

85. Shetty, A., Shetty, A. D., Pai, R. Y., Rao, R. R., Bhandary, R., Shetty, J. & Dsouza, K. J. (2022). Block Chain Application in Insurance Services: A Systematic Review of the Evidence. *SAGE Open, 12*. doi:https://doi.org/10.1177/21582440221079877

86. Siemens. (n.d.). *2007–2020: Defining digitalization*. Retrieved January 3, 2023, from Siemens: https://www.siemens.com/global/en/company/about/history/company/2007-2018.html

87. Talesra, K., & Nagaraja, G. S. (2021, May). Low-Code Platform for Application Development. *International Journal of Applied Engineering Research, 16*, 346-351. doi:10.37622/IJAER/16.5.2021.346-351

88. TechTarget. (2022, April). *Compiler*. Retrieved January 22, 2023, from TechTarget: https://www.techtarget.com/whatis/definition/compiler

89. Torres, M. (2021, May 12). *ServiceNow named a Leader in Low-Code Development Platforms*. Retrieved October 16, 2022, from ServiceNow: https://www.servicenow.com/blogs/2021/leader-in-low-code-development-platforms.html

90. Tozzi, C. (2021, March 1). *A practical take on low-code vs. traditional development*. Retrieved October 15, 2022, from TechTarget: https://www.techtarget.com/searchsoftwarequality/tip/A-practical-take-on-low-code-vs-traditional-development

91. Understand Insurance. (n.d.). *Household insurance*. Retrieved October 22, 2022, from Understand Insurance: https://understandinsurance.com.au/types-of-insurance/household-insurance

92. Volosovych, S., Zelenitsa, I., Kondratenko, D., Szymla, W., & Mamchur, R. (2021, January). Transformation of insurance technologies in the context of a pandemic. *Insurance Markets and Companies, 12*, 1-13. doi:10.21511/ins.12(1).2021.01

93. Waja, G., Shah, J., & Nanavati, P. (2021, April). Agile software development. *International Journal of Engineering Applied Sciences and Technology, 5*. doi:10.33564/IJEAST.2021.v05i12.011

94. Your Europe. (2022, February 11). *Insurance products*. Retrieved November 14, 2022, from https://europa.eu/youreurope/citizens/consumers/financial-products-and-services/insurance-products/index_en.htm

APPENDICES

Appendix 1: Povzetek (Summary in Slovene language)

V današnjem svetu tehnološki razvoj napreduje z veliko hitrostjo in poenostavlja življenja ljudi. Novosti na trgu se pojavljajo vse pogosteje kot prej in spodbujajo podjetja , da sledijo najnovejšim trendom na trgu, da bi bila pred konkurenco. Ena izmed teh novosti, ki je v zgodnji fazi priljubljenosti, je malokodno razvojno okolje (LCDP).

LCDP-ji so platforme, ki razvijalcem in ljudem z malo izkušnjami v razvoju omogočajo razvoj aplikacij z minimalnim pisanjem kode. Njihova glavna prednost je učinkovitost izdelave delujoče aplikacije, ki omogoča večjo produktivnost, nižje stroške, lažje vzdrževanje aplikacij in vključevanje deležnikov v razvojni proces (Talesra & S., 2021). Zaradi teh prednosti so LCDP-ji postali zelo privlačna rešitev za številne zavarovalnice, še posebej, ker so zaradi digitalizacije mnogi sistemi zavarovalnic zastareli in jih je težko vzdrževati. Z uporabo LCDP-jev se zmanjša odvisnost zavarovalnic od visoko usposobljenih razvijalcev. To pomeni, da bi lahko poslovni uporabniki iz zavarovalnic ustvarili nove zavarovalniške produkte ali spremenili obstoječe z ali brez pomoči razvijalca, s čimer bi naredili velik prostor za inovacije, ki jih je mogoče ustvariti v kratkem času (Daly, 2020).

Na trgu LCDP-jev za podjetja obstaja veliko podjetij, ki tekmujejo, katero podjetje bo ponudilo najboljši LCDP. Kljub podobnosti v arhitekturi in procesih platform je tržni fokus dobaviteljev lahko drugačen – nekatere so podjetniške platforme, druge pa temeljijo na določeni industrijski panogi. Podjetniške platforme so splošne platforme, ker jih je mogoče uporabiti za izdelavo aplikacij za katero koli industrijo. V magistrski nalogi bom navedla prednosti in slabosti dveh LCDP-jev. Prva platforma je Mendix, ki velja za splošno platformo za izdelavo kakršne koli aplikacije. Obstajajo trije glavni razlogi, zakaj sem za analizo v svoji magistrski nalogi izbrala Mendix. Prvič, ta platforma je umeščena med vodilne z najvišjo celovitostjo vizije v čarobnem kvadrantu za podjetniške LCPD-jev Gartner za leto 2020 (Gartner, 2020). Drugič, Mendix je objavil veliko primerov uporabe Mendixa s strani zavarovalnic. Tretjič, Mendix platforma je bila uporabljena s strani podjetja Adacta, ki je razvilo platformo AdInsure – drugi LCDP, izbran za analizo. Mendix plaforma je bila uporabljena za razvoj mobilne aplikacije za zavarovalniško ponudbo, in je bila integrirana s platformo AdInsure, kar potrjuje združljivost Mendixa z zavarovalniškim sektorjem.

AdInsure, ki predstavlja drugi LCDP, ima malokodno orodje, ki temelji na grafičnem uporabniškem vmesniku (GUI), imenovano AdInsure Studio. Poslovnim uporabnikom in IT strokovnjakom v zavarovalnicah omogoča vpeljavo sprememb v konfiguracijo zavarovalniških procesov in produktov.

Razlog za vključitev Adinsure Studia v analizo so moje dosedanje izkušenje pri delu z njim in zaradi enostavnega dostopa do virov, potrebnih za analizo. LCDP-ji, specializirani za posamezne panoge, niso brezplačni, zaradi česar so težko dostopni.

Namen magistrske naloge je prispevati k razumevanju LCDP-jev in njihove uporabe v zavarovalništvu. Glavni cilj magistrskega dela je analizirati funkcionalnosti industrijskih in splošnih LCDP-jev, jih primerjati in poiskati glavne razlike preko razvoj poslovne programske rešitve , imenovane Zavarovanje d oma v izbranih platformah.

Zavarovanje doma je vrsta zavarovanja, ki krije izgubo in škodo, povzročeno na nepremičnini zaradi škodnega dogodka. Finančna zaščita je povezana z zavarovanim predmetom, ki je stavba in lahko zajema tudi prizidane zgradbe, kot je garaža, osebne stvari v zgradbah, stroške, povezane s škodo osebe, ki je nastala na zavarovani nepremičnini, kot je poškodba in dodatni življenjski stroški, ki so nastali zaradi izgube ali poškodbe zavarovanega premoženja. Zavarovanje doma lahko kupijo posamezniki, ki so lastniki hiše, najemajo nepremičnine in najemodajalci (Understand Insurance, n.d.). Oseba, ki sklene zavarovanje doma se imenuje zavarovanec. Pogoji v zvezi s paketom zavarovanja so določeni v zavarovalni polici, ki predstavlja pravno zavezujočo pogodbo med zavarovalnico in zavarovancem. Plačilo za zavarovanje doma se imenuje premija in je zavarovanec dolžan plačati zavarovalnici (Fabozzi & Drake, 2010). Na višino zavarovalne premije vpliva veliko dejavnikov, ki so povezani z verjetnostjo nastanka škodnega dogodka. Če je verjetnost večja, bo višji tudi znesek premije (Dorfman, 1998). Za razvoj produkta Zavarovanje doma sem se odločila zaradi razpoložljivosti podatkov o vseh izračunih v zvezi z zavarovalno premijo.

Glavna osnova za primerjavo bo implementacija definiranih poslovnih zahtev v poslovno programsko rešitev na obeh platformah. Poslovne zahteve za razvoj zavarovalniškega produkta so pridobljene s poslovno analizo potreb zavarovalnice, ki ponuja zavarovanje doma Vsebujejo splošne informacije o produktu ter podrobnosti o izračunu kritja, poslovni proces, pogodbene udeležence in informacije o uporabniškem vmesniku, ki so pogosti vnosi za vse zavarovalne produkte. Rezultati analize bodo uporabljeni za opredelitev, katere funkcionalnosti potrebuje zavarovalnica za implementacijo in vzdrževanje omenjenega zavarovalnega produkta po malokodnem načelu.

Cilji:

- Opredeliti LCDP-je , njihovo splošno arhitekturo in njihov razvojni proces.
- Določiti razvojne specifikacije programskih rešitev.
- Ugotoviti ključne funkcionalnosti LCDP-jev, ki so potrebne za razvoj zavarovalniškega produkta.

Raziskovalno vprašanje: Katere so ključne razlike med uporabo industrijsko usmerjenih in splošnih LCDP-jev za razvoj zavarovalniške rešitve, na podlagi primerjave platform Mendix in AdInsure?

Magistrsko delo vsebuje teoretični in praktični del. V prvih treh poglavjih so predstavljeni teoretični del, kjer s pomočjo sekundarnih virov opredelim LCDP-jev, glavne zavarovalne pojme, ki jih bom uporabljala v praktičnem delu, trende v zavarovalništvu ter opis analiziranih LCDP-jev. V praktičnem delu bom s pomočjo primarnih virov naredila primerjalno analizo izbranih LCDP-jev z implementacijo definiranih poslovnih zahtev za programsko rešitev Zavarovanje doma . Implementacija produkta bo izvedena na vsaki platformi, njegove prednosti in slabosti pa bodo analizirane v vsakem ciklu razvojnega procesa. Na koncu bom identificirala potrebne funkcionalnosti LCDP-ja za popoln razvojni cikel izbranega zavarovalniškega produkta.

Ta raziskava je pokazala, da je možno zavarovalniški produkt uspešno razviti v splošnih in zavarovalniških LCDP-jih. Zavarovalniški LCDP-ji ponujajo funkcionalnosti z zavarovalniškim kontekstom in boljše urejevalnike za poslovna pravila. Splošni LCDP-ji ponujajo zmogljivejše urejevalnike uporabniškega vmesnika, močno skupnost in splošne funkcionalnosti, ki jih je mogoče uporabiti za konfiguracijo zavarovalniških rešitev.

Kljub razlikam v funkcionalnostih, je razvoj produkta Zavarovanje doma v dveh različnih LCDP-jih potrdil njihove glavne prednosti: hitrejši razvoj in nižje stroške. Rešitev je bila razvita za tri uporabnike v vsaki platformi: administrator, agent in stranka. Namen produkta je strankam ponuditi informativen izračun premije, ki vključuje izbrana kritja. Vendar je ustvarjanje in izdajanje ponudbe le en kos sestavljanke. Zavarovalnice imajo računovodske postopke, fakturiranje, terjatve, upravljanje polic, prodajo in pozavarovanje. Vsi ti procesi niso ustvarjeni in digitalizirani na eni platformi. AdInsure kot zavarovalniška platforma ponuja in podpira vse te procese, razdeljene na module. Nova konfiguracija, ki jo je izdelal AdInsure Studio, je strukturirana tako, da je kompatibilna z vsemi zavarovalniškimi moduli. Kot je bilo že rečeno, zavarovalnica lahko potrebuje samo en modul za implementacijo v svoj sistem in ne vseh modulov. V Mendixu lahko uporabnik razvije aplikacijo za katero koli industrijo. Njihov fokus ni na določeni industriji; osredotočeni so na ponudbo širšega nabora funkcionalnosti, da bi zadovoljili modeliranje aplikacij v vsaki industriji. Zavarovalnice imajo zapletene module in splošni LCDP-ji niso vedno primerni za razvoj celotnega zavarovalniškega sistema.

Po mojem mnenju bi morale zavarovalnice razmisliti o moči združevanja zavarovalniških in splošnih LCDP-jev. Splošni LCDP-ji so zmogljivejši za izdelavo aplikacij s sodobnim uporabniškim vmesnikom, ki so primerni za vse naprave, ter ciljajo na prvi stik s potencialnimi strankami. V kombinaciji z API-ji iz zavarovalniških LCDP-jev bi lahko te aplikacije razvili hitro, kar bi podjetju dalo veliko prednost. Bistveno je omeniti, da vsi splošni in zavarovalniški LCDP-ji nimajo enakega spektra funkcionalnosti. Tudi LCDP-ji, uvrščeni v isto skupino, kot so zavarovalniški ali splošni LCDP-ji, se razlikujejo. Izbira splošnega in zavarovalniškega LCDP-ja je za podjetje pomembna naložba, zato je potrebna predhodna analiza trga. Podjetja bi se morala osredotočiti na iskanje najboljše kombinacije platform, ki bi jim pomagala razviti sodobne in robustne rešitve, ki zadovoljujejo potrebe njihovih uporabnikov.

Appendix 2: Adapted business requirements and premium calculation for Home insurance

Building attributes

Attribute	Attribute type	Values	Mandatory
Building type	List	House, Condominium	true
Postal code	Integer		true
City	Text		true
Street	Text		true
House number	Text		true
Located in residential area	Boolean		true
Construction year	Integer		true
Building material	List	Brick, stone	true
Total area	Number		true
Usage type	List	Own usage Leased	true
Building condition	List	Adequate Inadequate	true
Inhabited building	Boolean		true
Number of occupants	Integer		true

Risks

Rule number	Input	Output
	buildingSumInsured	riskCode
1.	!=0	"Fire", "Explosion", "Lightning", "Storm", "Burglary", "Robbery", "Earthquake"

Coverage options

Rule number	Input	Output
1.	householdContents === true && householdContentsSum != 0	householdContentSum
2.	highValueContents === true && highValueContentsSum != 0	highValueContentsSum
3.	artifacts === true && artifactsSum != 0	artifactsSum
4.	jewelry === true && jewelrySum != 0	jewelrySum
5.	cash === true && cashSum != 0	cashSum
6.	liability === true && liabilitySum === “Bronze”	10000
7.	liability === true && liabilitySum === “Silver”	20000
8.	liability === true && liabilitySum === “Gold”	30000
9.	liability === true && liabilitySum === “Bronze”	40000

Base premium rate

Rule number	Input	Output
	buildingType	***Base premium rate***
1.	Condominium	flatBasePremium = 0.09
2.	House	detachedHouseBasePremium = 0.12

Multipliers

Rule number	Input		Output	
	attribute	**value**	**value**	**code**
1.	usageType	"Own usage"	1	usageTypeMultiplier
2.	usageType = "Leased Out"	"Leased Out"	1.2	usageTypeMultiplier
3.	usageType = "Leased"	"Leased"	1.4	usageTypeMultiplier
4.	buildingNumberOfOccupants	>4	1.5	numberOfOccupantsMultiplier
5.	buildingIsInhabited	false	1.6	isInhabitatedMultiplier
6.	locatedInResidentialArea	false	1.8	locatedInResidentialAreaMultiplier

Total multiplier:

usageTypeMultiplier * numberOfOccupantsMultiplier * isInhabitatedMultiplier * locatedInResidentialAreaMultiplier

Modifiers

Rule number	Input	Output	
	buildingType	**coverageOption**	**Modifier**
1.	"Building"		1.0
2.	"Outbuilding"		1.1
3.		"HouseholdContents"	1.2
4.		"HighValueContents"	1.6
5.		"Artifacts"	2.0
6.		"Jewelry"	3.3
7.		"Cash"	2.5
8.		"Liability"	5.0

Premium rate per coverage

Rule	Input	Output	
	buildingType	Code	Value
1.	"Condominium"	buildingRate	flatBasePremium * modifier
2.	"House"	buildingRate	detachedHouseBasePremium * modifier
3.		householdContentsRate	flatBasePremium * totalMultiplier * modifier
4.		highValueContentsRate	flatBasePremium * totalMultiplier
5.		artifactsRate	flatBasePremium * totalMultiplier * modifier
6.		jewelryRate	flatBasePremium * totalMultiplier * modifier
7.		cashRate	flatBasePremium * totalMultiplier * modifier
8.		liabilityRate	flatBasePremium * totalMultiplier * modifier

Premium per coverage

Rule	Coverage	Calculation
1.	Building	buildingSum * buildingRate
2.	HouseholdContents	householdContentsSum * householdContentsRate
3.	HighValueContents	highValueContentsSum * highValueContentsRate
4.	Artifacts	artifactsSum * artifactsRate
5.	Jewellery	artifactsSum * artifactsRate
6.	Cash	cashSum * cashRate
7.	Liability	liabilitySum * liabilityRate

Total premium = Sum of all premiums per coverage