



INSTITUTO
UNIVERSITÁRIO
DE LISBOA

On the Low-Code Premise: OutSystems and Mendix for Real-World Enterprise Applications

Diogo Filipe Bento Prata Pina

Master in Telecommunications and Computer Engineering,

Supervisor:

PhD Pedro Joaquim Amaro Sebastião, Associate Professor with
Aggregation,
ISCTE-IUL

October, 2024

Department of Information Science and Technology

On the Low-Code Premise: OutSystems and Mendix for Real-World Enterprise Applications

Diogo Filipe Bento Prata Pina

Master in Telecommunications and Computer Engineering,

Supervisor:

PhD Pedro Joaquim Amaro Sebastião, Associate Professor with
Aggregation,
ISCTE-IUL

October, 2024

Acknowledgments

As I conclude my dissertation, I would like to extend my heartfelt thanks to those who provided unwavering support throughout this whole journey.

I would like to thank my supervisor Professor Doutor Pedro Sebastião, for all his supervision, availability, support and guidance during my investigation and writing, and most importantly I would like to thank him for trusting in me to embrace this project.

I would also like to extend my gratitude to all my Low-Code colleagues who provided invaluable support, helping me getting the correct information throughout my research and implementation processes.

Lastly, but most important I would like to thank my family and my girlfriend for always standing by my side during the times I needed them most, their constant support and motivation were crucial and helped me to complete this journey.

"Success is not final, failure is not fatal: it is the courage to continue that counts."

-Winston Churchill

[This page has been intentionally left blank for formatting purposes]

Resumo

Durante muitos anos, a maioria das empresas dispunha de poucas opções para o desenvolvimento de aplicações: podiam comprar soluções de fornecedores externos ou desenvolver aplicações personalizadas de raiz.

O aparecimento e contínuo crescimento de diversas plataformas Low-Code/No-Code (LCNC) permitiram que empresas e utilizadores tomassem controlo do processo de desenvolvimento de software. Assim sendo, nos últimos anos, diversas plataformas estabeleceram-se firmemente como ferramentas essenciais em várias organizações.

Métricas como usabilidade, redução de custos, rápida integração e processamento de dados bem como desenvolvimentos mais céleres tornaram-se indispensáveis para a maioria das empresas que no cenário atual onde reina a complexidade, estão sob pressão constante para entregar aplicações de forma mais rápida enquanto simultaneamente elevam os padrões de qualidade e flexibilidade.

Foi esta mudança, que tornou estas plataformas mais atrativas, pois, através das mesmas, as empresas podem desenvolver aplicações com menos utilização de recursos técnicos, dando resposta às constantes variações nos mercados, facilitando a competitividade das empresas nos seus ecossistemas.

Por conseguinte, esta dissertação visa explorar diferentes plataformas low-code, realizando um estudo detalhado das mais proeminentes, sendo elas Mendix e OutSystems.

O estudo atual baseia-se na análise das funcionalidades das mesmas, focando-se também numa comparação mais aprofundada entre elas, incluindo a sua usabilidade, potenciais custos das plataformas e a forma como se estão a inserir no ainda recente mundo da Inteligência Artificial.

Pretende-se por fim avaliar como é que plataformas como a Mendix e a OutSystems podem ser adotadas de forma eficaz pelas empresas para melhorar as suas operações.

Palavras-Chave: Low-Code, OutSystems, Mendix, Agile, Aplicações Empresariais, Inteligência Artificial

[This page has been intentionally left blank for formatting purposes]

Abstract

For many years, most businesses and companies did not have many routes to take regarding application developments, they could either purchase new solutions from third-party vendors or build custom applications from the ground up with a team of skilled developers.

However, the emergence and increasing maturity of low-code/no-code (LCNC) platforms have empowered users throughout organizations to take control of the application development process themselves and so, over the past few years, low-code platforms have established themselves as essential tools in numerous organizations and businesses.

Terms like easy-use, lower costs, smooth data integration and faster development have become a must for most enterprises, and in today's increasingly complex business environment, organizations are under constant pressure to deliver applications faster while raising the quality and flexibility stakes.

This increasingly sought-after shift has made low-code platforms more attractive, since, through these platforms, businesses can rapidly develop applications with less technical resource utilization. Secondly, these platforms will allow quick responses to changing market conditions, thus further enabling companies to stay competitive in the ecosystems.

Therefore, the goal of this dissertation is to explore low-code platforms, going through a detailed study of the most prominent ones, Mendix and OutSystems.

The current study is based on the feature analysis of these platforms, also focusing on a deeper comparison between them, including their usability, study and also examine the possible costs and newly developed AI tools they offer. It aims to assess how effectively Mendix and OutSystems can be adopted by companies to improve their operations.

Keywords: Low-Code, OutSystems, Mendix, Agile, Enterprise Software, Artificial Intelligence

[This page has been intentionally left blank for formatting purposes]

Table of Contents

ACKNOWLEDGMENTS	I
RESUMO.....	III
ABSTRACT	V
TABLE OF CONTENTS	VII
LIST OF FIGURES	IX
LIST OF TABLES.....	XI
ACRONYMS	XIII
CHAPTER 1 - INTRODUCTION.....	1
1.1. MOTIVATION AND CONTEXT	1
1.2. RESEARCH QUESTIONS.....	2
1.3. RESEARCH METHODOLOGY.....	3
1.4. DISSERTATION STRUCTURE	3
1.5. DISSERTATION GOALS.....	4
1.6. RESEARCH CONTRIBUTIONS	4
CHAPTER 2 - STATE OF THE ART	5
2.1. TECHNOLOGY BACKGROUND OVERVIEW	5
2.2. SOFTWARE DEVELOPMENT METHODOLOGIES	6
2.2.1. <i>The Waterfall Model</i>	6
2.2.2. <i>The Spiral Model</i>	8
2.2.3. <i>Rapid Application Development (RAD)</i>	11
2.2.4. <i>Dynamic System Development Method</i>	13
2.2.5. <i>Rational Unified Process (RUP)</i>	18
2.2.6. <i>Agile Methodology</i>	20
2.3. THE GROWTH OF LOW-CODE TECHNOLOGIES	25
2.4. LOW-CODE PLATFORMS USE-CASES.....	28
CHAPTER 3 - LOW-CODE APPLICATION PLATFORMS ANALYSIS.....	31
CHAPTER INTRODUCTION	31
3.1. LEADING LOW-CODE PLATFORMS	31
3.2. OUTSYSTEMS.....	34
3.2.1. <i>OutSystems Cloud and OutSystems 11 Platform Architecture</i>	35
3.2.2. <i>ODC Cloud-Native Architecture</i>	38
3.3. MENDIX	40
CHAPTER 4 - TECHNICAL, COST, AND AI COMPARISON.....	45
CHAPTER INTRODUCTION	45
4.1. ARCHITECTURE & INTERFACE	46
4.2. DATABASE & ENTITY RELATIONSHIP.....	48
4.3. COMMUNITY SUPPORT	52
4.4. ERROR HANDLING	52
4.5. INTEGRATIONS	56
4.6. PRICING PLANS.....	63
4.6.1. <i>OutSystems pricing model</i>	63

4.6.2.	<i>Mendix pricing model</i>	67
4.7.	AI-ASSISTED DEVELOPMENT (AIAD)	70
4.7.1.	<i>OutSystems AI Tools</i>	70
4.7.2	<i>Mendix AI Tools</i>	72
CHAPTER 5 - CONCLUSIONS AND FUTURE RESEARCH		75
5.1	CONCLUSIONS	75
5.2.	FUTURE RESEARCH	76
REFERENCES		77
APPENDIX		87
APPENDIX A		87
APPENDIX B		88
APPENDIX C		89
APPENDIX D		90
APPENDIX E		91
APPENDIX F		93
APPENDIX G		95
APPENDIX H		97

List of figures

Figure 1 - Representation of Royce's first Waterfall model from 1970	7
Figure 2 - The Spiral Model of Software process	9
Figure 3 - The 4 phases of RAD	12
Figure 4 - DSDM's fundamental principles.....	14
Figure 5 - DSDM lifecycle phases	16
Figure 6 - Rational Unified Process Whale Chart	18
Figure 7 - Agile Software Development Lifecycle.....	23
Figure 8 - Main reasons for using LCDP according to IT Users	26
Figure 9 - Main reasons for not using or considering low-code development platforms	27
Figure 10 - Gartner's Magic Quadrant for Enterprise Low-Code Application Platforms	32
Figure 11 - Gartner's Magic Quadrant structure explained	32
Figure 12 - OutSystems application lifecycle management components	35
Figure 13 - OutSystems Service Studio common use case.....	36
Figure 14 - OutSystems Integration Studio .NET extension use case	36
Figure 15 - OutSystems running on a private cloud.....	37
Figure 16 - OutSystems running on-premises.....	38
Figure 17 - Mendix Software Development Life Cycle representatio	40
Figure 18 - Key Components of the Mendix Platform Architecture	42
Figure 19 - OutSystems vs Mendix: Core Messages.....	45
Figure 20 - Mendix Studio Pro IDE	46
Figure 21 - Different database options available at the application configuration settings	49
Figure 22 - Design of a scalable data model for E-commerce Orders and Reviews using Mendix	51
Figure 23 - Design of a scalable data model for E-commerce Orders and Reviews using OutSystems	52
Figure 24 - Error handling options for Microflows and Nanoflows in Studio Pro	53
Figure 25 - TrueChange technology live working in OutSystems IDE	54
Figure 26 - User-Exception use-case scenario.....	55
Figure 27 -Request body of the Post Completion method defined in the exposed REST API using OutSystems.....	59
Figure 28 - Request body of the Post Completion method in the exposed REST API using Mendix	59
Figure 29 - User roles defined in OutSystems and Mendix respectively	60
Figure 30 - DS_CallRest_Post microflow logic.....	62
Figure 31 - ACT_PostCompletions microflow logic	62
Figure 32 - Different pricing structure for a "one app" license	67
Figure 33 - Different pricing structure for multiple apps license	67
Figure 34 - AI Mentor system engine	70
Figure 35 - Maia chat interface used in Studio Pro	72
Figure 36 - Best Practices Recommender Pane	73
Figure 37 - Logic Recommender dialog box	73
Figure 38 - Workflow Recommender dialog box	74

[This page has been intentionally left blank for formatting purposes]

List of tables

Table 1 - Analysis of the Strengths and Weaknesses of the Waterfall Model	8
Table 2 - Analysis of the Strengths and Weaknesses of the Spiral Model	11
Table 3 - Strengths and weaknesses of RAD methods.....	13
Table 4 - Analysis of the Strengths and Weaknesses of the DSDM Model	17
Table 5 - Analysis of the Strengths and Weaknesses of the RUP Model	20
Table 6 - Integration Use-Case findings comparison	63

[This page has been intentionally left blank for formatting purposes]

Acronyms

AI – Artificial Intelligence

AIAD - AI-Assisted Development.

AOs – Application Objects

aPaaS - Application Platform as a Service

B2C – Business-To-Consumer

B2E – Business-To-Employee

CDN - Content Delivery Network

CI/CD – Continuous Integration and Continuous Delivery

CRM - Customer Relationship Management

DSDM - Dynamic Systems Development Method

ERD – Entity-Relationship Model

GenAI – Generative Artificial Intelligence

GUI - Graphical User Interface

HSQldb - HyperSQL DataBase

HTTP - Hypertext Transfer Protocol

IaaS - Infrastructure as a Service

ICT – Information and Communication Technology

IT – Information Technology

IDC – International Data Corporation

JSON - JavaScript Object Notation

JVM – Java Virtual Machine

KKR - Kohlberg Kravis Roberts

LCDP – Low-Code Development Platforms

LCNC – Low Code/No Code

MAIA – Mendix Artificial Intelligence Assistance

NATS – Networking Asynchronous Transfer System

OCI – Oracle Cloud Infrastructure

ODC - OutSystems Developer Cloud

PaaS - Platform as a Service

RAD – Rapid Application Development

RDS - Relational Database Service

RAD – Rapid Application Development

REST - Representational State Transfer

RUP – Rational Unified Process

SaaS – Software as a Service

SDLC – Software Development Lifecycle

TCO – Total cost of ownership

UI – User Interface

URL - Uniform Resource Locator

WAF – Web Application Firewall

XML - eXtensible Markup Language

Introduction

1.1. Motivation and Context

In the fast-changing world of software development, innovation has always been the key factor behind progress and amid the many technological advancements that have changed the way we build software low-code platforms stand out as a game-changing approach [1].

The high demand for technology and new applications combined with the lack of digital talent created the perfect scenario for the emerging appearance of Low-Code. This approach minimizes the need of manual coding and in some cases, it even eliminates it completely (no-code platforms), along with other features like a user-friendly drag-and-drop interface and the allowance to reuse modules and components in other formats like, web and mobile, consequently accelerating the development time, improving efficiency and reducing maintenance. As a result of these time and cost improvements, companies faced a reduction of nearly 74% in the cost of applications [2].

Also pumped by global events like the Covid-19 pandemic and Brexit, the information and communication technology sector (ICT) had a high demand for new applications and as evidence the IDC estimated that by the end of 2023 the number of new applications that will need to be created globally would reach the 500 million, and that would surpass the number of new applications created in the last 40 years [3]. Consequently, Low-Code platforms dimension rise as they are used as a way to address the shortage of skilled digital professionals, to maximize the productivity of businesses resources and to fulfil the market demands on delivering more faster, more agile and more secure applications [2].

It is relevant to notice but not unexpected that the LC/NC industry is also expanding since a long time ago, and is aiming for notable forecasts and figures, that can firmly demonstrate its relevance in the ICT sector. Gartner projected the LCNC market to reach \$26.9 billion by 2023, while Forrester offers an even higher estimate, predicting it will hit \$45.5 billion by 2025. Additionally, by 2024, it's anticipated that 65% of app development will utilize low-code approaches, reflecting the swift adoption of these platforms [4].

When it comes to low-code development, it is not possible to discuss the evolution of low-code platforms without mentioning two of the industry's leading giants: Mendix and OutSystems. OutSystems, initially founded in Portugal and now based in the United States, and Mendix, headquartered in the Netherlands but acquired by German industrial powerhouse Siemens in 2018, have achieved what many software companies aspire to: becoming globally recognized leaders in the enterprise low-code software market. They have consistently outperformed their competitors and continue to raise the bar for what enterprise low-code solutions can deliver [5].

More recently, with the emergence of AI, GenAI tools are becoming a staple in software development [6] and synergies between two major forces (LC/NC platforms and generative AI) were bound to happen. This indeed means that the possible partnership of LCNC platforms and generative AI is a quite promising future in software development. The integration of gen AI into low-code platforms might result in streamlined development, empowered non-technical users, efficiency, and innovation taken to the next level. The results that might come from this, hold immense promise across industries for quicker time-to-market, high-quality software, and business agility.

1.2. Research Questions

The research questions that drove me into writing this master thesis were:

- What are the benefits of adopting low-code strategies to enhance businesses?
- What is the impact of adopting a low-code tool in terms of cost, time, and quality compared to using a traditional development tools and methodologies?
- How can platforms like Mendix or OutSystems accelerate digital transformation in businesses?
- How can organizations select the most suitable low-code platform based on their needs and application types?
- How do low-code platforms integrate with generative AI to enhance software development processes?
- What role do low-code platforms play in helping businesses quickly adapt to the growing demands for AI-driven solutions?

1.3. Research Methodology

The research approach used in this dissertation is a combination between a Comparative Analysis that will be employed in this dissertation as a means to study the transformative role of Low-Code platforms and a Market Research where the trends, the market demand and adoption rates regarding Low-Code have been interpreted with the help of industry reports and market data analysis.

Both these methods have been supplemented by a SWOT Analysis along the paper with the aim to show the Strengths, Weaknesses, Opportunities, and Threats for each identified platform.

Finally the Comparative Analysis performed for the necessary technical characteristics, pricing models, and pitched AI integrations supports two use cases: one related to data modelling capabilities regarding e-commerce order management and one other related to integration via REST APIs for an AI chatbot.

1.4. Dissertation Structure

The structure of this dissertation is organized into 5 distinct chapters:

- **Chapter 1:** This chapter describes the recent proliferation of Low-Code platforms, which has been driven by rising demand for digital solutions and a shortage of developer talent. It also presents the enunciations of research questions, objectives, and contributions of this study.
- **Chapter 2:** This chapter describes the state of the art based on an immersive technology background overview on both traditional and modern methodologies for software development, including Waterfall, Agile, and Low-Code platforms outlining the evolution of such methods.
- **Chapter 3:** The third chapter provides a more in-depth look at the two leading LC/NC platforms, OutSystems, and Mendix, focusing on their architecture, development tools, and components of each, in detail.
- **Chapter 4:** This chapter talks about technical and cost comparisons between OutSystems and Mendix. This section describes pricing models, inclusions of AI-powered development tools in both, presented here are use-cases, which offer insight into comparing capabilities in data modelling and integration scenarios.
- **Chapter 5:** Chapter 5 presents the conclusion and future works, summarizing the key findings obtained along the paper and pointing to further research efforts, such as advanced AI tools and a comparison of other leaders in the LC/NC market.

1.5. Dissertation Goals

The primary goal of my dissertation is to explore the transformative potential of LC/NC development platforms technologies. By investigating how these innovations challenge the traditional software development process, which relies on complex coding and specialized programmers.

I aim to demonstrate how LC/NC market leaders like Mendix and OutSystems can revolutionize the field. My research will focus on how these tools can make software development faster, more accessible, and inclusive. I will analyse their impact from both technical and business perspectives, assessing how they enhance agility, foster innovation, and democratize technology.

Additionally, I will examine both the architectures and development life cycle to understand how we can also use these tools to improve automation and creativity.

Through this research, I seek to highlight how these advancements drive digital transformation, increase development efficiency, and enable broader participation in the software creation process. Ultimately, my goal is to contribute to a future where software development is both simpler and more innovative.

1.6. Research Contributions

The continuous contributions of this dissertation, includes its contribution to the preliminary basis for further academic publication. Throughout this research process, it became clear that the number of insights and comparisons present in the paper could represent potentially valid contributions to the field of software engineering as well as to the Low-Code development platforms themselves.

This reacted initially on the early stages of a draft for a publishable article. Although still a developing article, it attempts to encapsulate the key findings from this thesis into a solid analysis of leading Low-Code platforms.

The key findings that will be discussed: different technical and cost advantages of Low-Code platforms, their increasing role in digital transformation driven by AI, among other features.

Finally, I would like to contribute to the academic discussion of Low-Code platforms since it underlines my thesis work and its eventual contribution to inform the industry of the practices that could assist business and IT professionals in applying correct and informed decisions on the adoption of Low-Code technologies.

State of the art

2.1. Technology Background Overview

To understand how low-code gained its place in the software industry, it is necessary to look back several years, not just when the first low-code technologies emerged but to the roots of software development.

During the early 1960s to the early 1970s, the Waterfall model emerged as one of the earliest structured methodologies for software development [7]. Firstly, introduced by Winston.W.Royce the Waterfall model consisted in 7 stages (system requirements, software requirements, analysis, system design, coding implementation, testing, operations). Although he believed in this model, he also stated that this implementation is risky and invites failure [7].

The popularity of the usage of Waterfall methodology was also pumped in the 1980s due to the increase of projects that required high reliability and quality, such as defence, aerospace, and government systems. Nevertheless, the waterfall model started to face some challenges and criticism during the 1990s and early 2000s due to the appearance of new technologies, changing markets, and growing customer demands since it was rigid, slow, and incapable of accommodating the dynamic and unpredictable nature of software development during that era [8].

As a reaction to the Waterfall Model, the Rapid Application Development (RAD) notion presented by James Martin in his book took off in the 1990s when the notion of visually "assembling" desktop applications using tools like Visual Basic, Delphi and Oracle Forms became famous driven by the rapidly evolving landscape of digital products [9] setting the first steps to what would be called Agile methodologies.

Recognizing the limitations and pitfalls of Waterfall models, companies were looking for a more dynamic and adaptive approach to fulfil their project management requirements, so in 2001 Ken Schwaber and Jeff Sutherland both members of a 17-engineers group, founded the Agile manifesto that outlined their common principles on how a contemporary software development process should function, aiming for flexibility and adaptability reckoning the basis that a developer does not need to define all the answers up front, as it is done in the waterfall models. Instead, he breaks the problem into digestible components [10].

Other methodologies also complemented Agile's principles over time like for example:

- **DevOps** : DevOps arose in the late 2000s in reaction to challenges that teams experienced using Agile across the full lifecycle of software, more specifically between development teams and operations. It touches every phase of the development and operations lifecycle at the best Agile practices: planning, building, monitoring, iterating. It brings together skills, processes, and tools from every facet of an engineering and IT organization [11].

- **CI/CD** : Continuous integration and continuous delivery automated the processes of building, testing, and deployment which was an essential aspect of CI/CD practices, enabling regular and reliable software releases [12].

Together, these complementary methodologies provided the foundation to integrate the most productive software development tools based on its main principles.

Such integration greatly amplified the market effect caused by the so-called new technologies, including LCNC development and AI-assisted development that use machine learning (ML) models.

2.2. Software Development Methodologies

Software development has significantly evolved over the past few decades, having its roots in Traditional Software Development Methodologies and later moving to Modern Software Development Methodologies. Such a transition showed the general trend in the industry for more adaptive, iterative, and collaborative approaches.

2.2.1. The Waterfall Model

The Waterfall model firstly introduced by Winston.W.Royce follows a linear path of development, characterized by fixed, orderly phases and when one ends, the other begins [13].

The model, which predated the formalization of software design, was presented as a means of controlling an increasingly complicated size of large-scale projects.

Basically, following a rigidly preordered sequence of phases with, in its original form there is no return to prior phases, waterfall, as the name would suggest, originally introduced by Dr. Winston W. Royce, has its process in a line with no iteration, meaning their process cannot revisit and revise any earlier stage and also extensive upfront planning is necessary, which cause delays in the early stages that will have an impact on all further phases [14].

Figure 1 represents the original model designed by Winston Royce introducing this 7 phases rule:

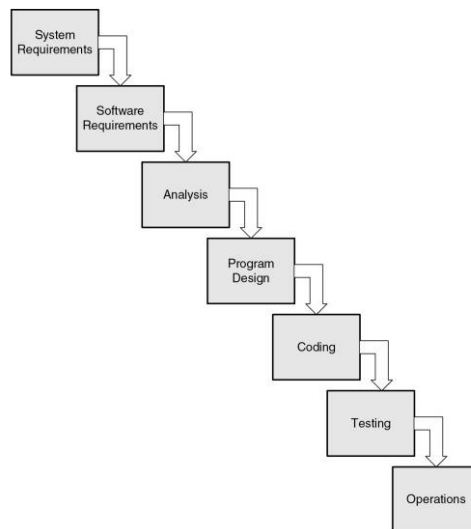


Figure 1 - Representation of Royce's first Waterfall model from 1970 (Source: [14])

The phases that collectively represent the Waterfall model [15]:

- **System requirements:** Identify all the system-level requirements—document them—defining what the system should be able to provide to the user.
- **Software requirements:** Detailed analysis and documentation of the software requirements derived from the system requirements, but this time focusing on what the software is going to do.
- **Analysis:** Involves breaking down and studying software requirements in-depth to sharpen the focus on project objectives, studying the problem areas, projecting outcomes, and creating a more detailed project plan.
- **Program Design:** The process of creating a clear software architecture that would be utilized in development as a sort of detailed blueprint. This involves algorithms and workflows.
- **Coding:** Transforms the design into actual source code. Developers follow predefined standards and specifications to write the code accordingly.
- **Testing:** This process systematically runs the software in different environments to verify that it meets the intended specifications and to identify any defects that need correction.

- **Operations:** The release of the software into a live environment, including installation and maintenance tasks. Post-deployment activities include monitoring to ensure stable and effective operation.

According to Barry.W Boehm “The Waterfall model has become the basis for most software acquisition standards” [16] with several advantages but at the same it involves notable risks and feasibility challenges. An analysis of both its strengths and weaknesses is presented in Table 1.

Table 1 - Analysis of the Strengths and Weaknesses of the Waterfall Model (*Source: [17]*)

Strengths	Weaknesses
- Due to its linearity, it's simple and easy to understand. It provides a direct path in development. - The emphasis given to planning and upfront documentation in the model; hence, is effective in projects that have considerable scope and are large. - Stages that can be set along with deadlines to enable orderly progress, thus helping in efficiently tracking a project [17].	- Due to the strict sequential order in which it carries out processes. The iteration development or adjustment gets forbidden. - Changing does indeed mean it is a pain to go back and revise earlier stages, which in turn can delay the projects themselves. - This model will work only on the premise of a landscape that is static and does not change, which in software development is not normal nor ideal [17].

2.2.2. The Spiral Model

Moving from the Waterfall model, another traditional software development methodology is the Spiral model, presented by Barry Boehm in 1986 [16], it extends the rigid sequential approach into iterative cycles of planning, risk analysis, engineering, and evaluation. This methodology will not only retain systematic structure but also provide room for adaptability through continuous project refinements, fitting well with the evolving project requirements.

In figure 2 it represented the original spiral model defined by Boehm with its four phases divided into different quadrants:

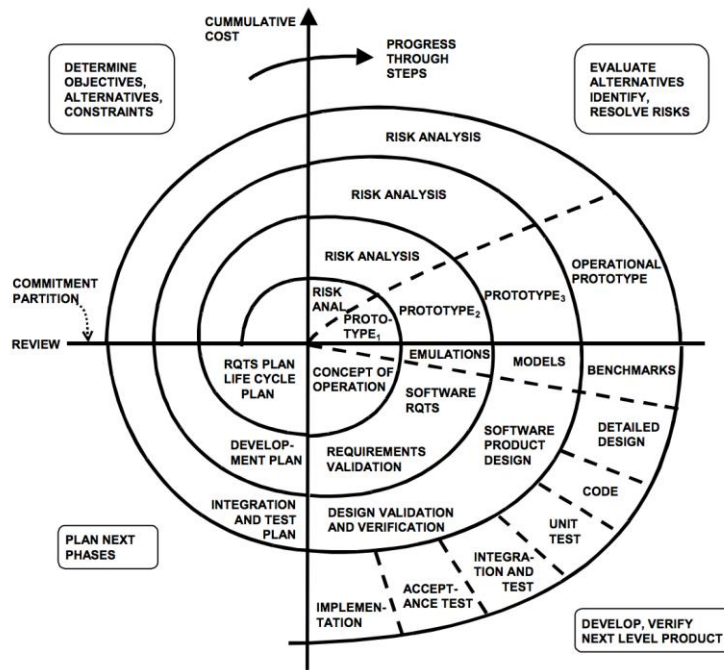


Figure 2 - The Spiral Model of Software process (Source: [16])

According to Boehm, every spiral model cycle should consider the following 4 phases:

- **Planning phase (defining objectives):** It is within this planning phase that the definition of project goals, requirements, and scope must be made, while risks are assessed. The attention should be on the alignment of user needs with stakeholder expectations, thereby laying a foundation for the whole project. Also, the stakeholder involvement will be very instrumental in elaborating the success criteria of the project, hence ensuring its alignment [18].
- **Risk analysis phase:** After defining project objectives, the team conducts a detailed risk analysis to uncover potential issues like technical hurdles, resource constraints, or changing requirements. Strategies are then developed to mitigate these risks, making the project more resilient. Both quantitative and qualitative methods are used to assess and prioritize risks effectively [18].

- **Product Development phase (Engineering):** After the risk analysis and strategic planning, the development phase starts, where the respective sequence of development involves coding, followed by integration and testing. This phase should be focused on the previously set objectives, evaluating ongoing results to determine if everything is on the right track and addressing any immediate problems. Customer feedback and initial prototypes guide the iterative refinement of the software [18].
- **Evaluation Phase (Plan Next Iteration):** During the evaluation phase, the software undergoes assessment to verify if it meets customer needs. This phase ensures the software aligns with user expectations. It also checks if the final product maintains high quality, before starting a new cyclic iteration [19].

The spiral model became famous since it was helpful in specific cases, for example when costs and risk evaluation become important for medium to high-risk projects also when long-term project commitment is unwise due to possible changes in economic priorities and users are not sure about their requirements and the requirements are complex, or when significant changes are expected, particularly involving research and exploration [20].

In recent years, there has been significant shift from traditional development, toward agile methodologies mostly due to the growing need for flexibility and faster delivery [21].

This shift to modern approaches has become essential since modern methodologies offer greater adaptability, collaborative processes, and iterative development to meet evolving user needs and market demands.

However, much like the other models, it comes with its own set of pros and cons that are addressed and analysed in table 2.

Table 2 - Analysis of the Strengths and Weaknesses of the Spiral Model

Strengths	Weaknesses
- Spiral Model is ideal for projects with many unknown risks, thanks to its risk analysis and management at each phase. - It is flexible in requirements since it accommodates changes in requirements even at later phases. - Spiral Model focuses on risk management to reduce uncertainty and potential impacts. - The improved quality of the model, since Multiple iterations in Spiral Model help enhance software quality and reliability [19].	- Spiral Model can be expensive to implement due to its iterative nature and the extensive risk management processes involved. - Risk analysis requires highly specialized skills. Without them it will be very hard to implement this model. - Estimating time is challenging because the number of phases cannot be determined at the beginning of the project. - The Spiral Model is generally not well-suited for smaller projects, as its detailed processes and phases can be overkill for simple or low-risk endeavours [20].

2.2.3. Rapid Application Development (RAD)

One of the first approaches of an Agile software development methodology is the rapid application development (RAD), developed by James Martin in the 1980s and officially presented in 1991 as a response to the failures of traditional methods like Waterfall [22].

Rapid application development is an Agile approach to software development, whereby less emphasis is laid on adhering to a strict plan, and instead focuses on ongoing software projects through user feedback. It, therefore, emphasizes low-cost rapid prototyping rather than high-cost planning.

Though often mistakenly considered a specific model, the idea behind Rapid Application Development (RAD) is that we derive value by treating our software projects more like clay than steel—meaning they can be tested and refined, rather than strictly following detailed planning and design phases typical of traditional development models [23].

The RAD which focuses on quick and iterative development, traditionally includes 4 main phases that are depicted in figure 3, and each one is designed to expedite the development process and engage user feedback continuously:

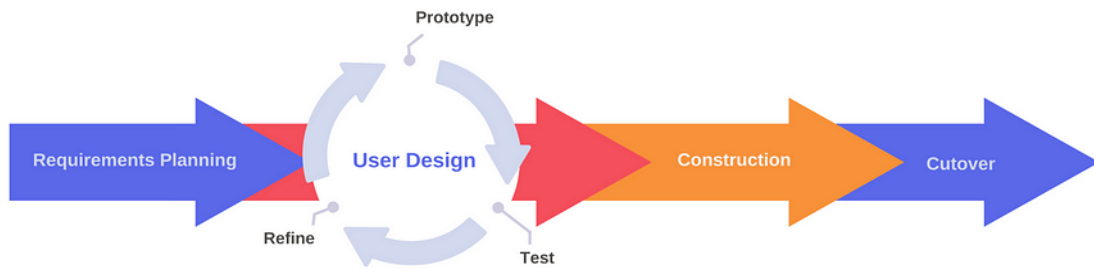


Figure 3 - The 4 phases of RAD (Source: [24])

- **Requirements Planning:** Instead of taking many months to create detailed specifications along with the users, RAD starts immediately with the definition of a flexible set of requirements. Flexibility is an inherent property of RAD, it supports the possibility of changing requirements at any time during the development cycle.
It essentially captures the essence from the developers while the client describes their vision for the product, leading to a mutual agreement with the developers on the requirements that correspond to this vision [23].
- **User Design:** The team works closely with users to create a prototype or mock-up of the software. This prototype is then used to gather feedback from users, allowing improvements to be made to the design before moving on to the next phase [24].
- **Rapid Constructions:** During this stage, the team builds the software product using tools and techniques specific to rapid application development. The prototype is used as a reference, and the product is created in incremental steps [24].
- **Cutover/Release:** The transition from development to the production environment involves final testing, deployment, and user training. During this phase, any remaining issues are thoroughly identified and resolved. Once testing is successful, the system is deployed, and users receive training to ensure they can use the software effectively [24].

It should also be emphasized that these phases are cyclical and may be revisited repeatedly until the final product satisfies the requirements and is prepared for deployment.

Rapid Application Development can also be seen as a predecessor to the more formal Agile methodologies we use today. Although RAD and Agile share common principles—like iterative development, user feedback, and flexibility [24], it remains unique in its own approach and characteristics with its own features and drawbacks, that are considered in table 3.

Table 3 - Strengths and weaknesses of RAD methods

Strengths	Weaknesses
- Decreased development time and enhanced efficiency with fewer team members result in reduced expenses. - The use of a time-box approach where each phase of development is allocated a fixed duration, helps control budget and timing uncertainties. - Continuous engagement of the customer throughout the project ensures alignment with client expectations and business objectives. - Employs various techniques to document and understand business operations, data flows, and processes which help in clarifying system requirements and design [26].	- The speed of development of RAD can lead to design shortcuts, potentially affecting the product's overall quality and functionality. - The RAD model requires experienced developers capable of handling rapid changes and decision-making to ensure project success. - May lead to a poorly designed system if emphasis is placed on rapid delivery rather than solid architectural planning. - While RAD shortens the time to market, it needs rigorous project management to ensure deliverables meet quality standards [25].

2.2.4. Dynamic System Development Method

Transitioning from the RAD model, defined by James Martin in 1991, into a Dynamic Systems Development Method within the software industry had a focus on a more structured approach, which did indeed incorporate speed and agility, albeit with an even stronger emphasis on project management and involvement of stakeholders. The Dynamic Systems Development Method (DSDM) is a framework that tries to bring speed to product delivery and guides through remaining in control throughout the whole process [27].

To fully understand how DSDM relates to the Agile School of methodologies, one must consider how the principles of DSDM support those core values held by the Agile development methodology.

Meant to be followed for a thorough execution of DSDM, disregarding the principles will weaken the whole and make the framework far less effective, raising project risks higher [28].

These principles that are depicted in figure 4 complete the model picture of DSDM:



Figure 4 - DSDM's fundamental principles (Source: [29])

- **Active user Involvement:** Active user participation is the first and most important principle. Users (who will be the user of the product) must be in sync with throughout the development. Continuous involvement of stakeholders makes the process less error-prone, which consequently reduces the cost for rectification. DSDM is based on the close working relationship of a small group of carefully chosen individuals using a language they all understand such that everyone involved in the development project is constantly communicating with each other [29].
- **Teams need to have the authority to make decisions:** Empowerment is a key tenet of the Dynamic Systems Development Method. This enables teams to take decisions fast during development, which speeds up the whole process. But that is no free-for-all, in the end, teams need to follow their guidelines within predetermined financial constraints, and more importantly they need to stick to the project scope. For instance, they may reinterpret a requirement of requirements in the context, adjust the priority of some tasks or solutions intended for the system [27].
- **Focus on frequent delivery of products:** High frequency for delivery of values to the customer ensures that the errors/bugs get identified, worked upon, and resolved/reversed/fixed at an early stage. Identification of the source of the error and the root cause is found and fixed. It applies to all-requirement documents, working models, as well as on program codes [28].

- **Fitness for business purpose is the essential criterion for acceptance of deliverables** In DSDM, there should be the delivery of software that covers the business need or requirement but which can be open for changes or enhancements in future iterations. DSDM has a greater emphasis on fulfilling the business requirement in the first place and will not invent or build unplanned or opportunistic software. It keeps the process uncomplicated and lean critically [29].
- **Iterative and incremental development necessary to converge on an accurate business solution:** The key essence of this approach is that it reduces a very large project into several feature-based projects of smaller scales. At each delivery, a new feature is introduced to the client. Such deliveries are done on an incremental basis till all features required by business are fully delivered [29].
- **All changes during development are reversible:** Adapting to change means that, at each increment development, the system configurations need to evolve, shifting priorities of requirements. In modern software tools, flexible configuration of projects according to this principle is allowed. Some are afraid that going back to a previous step can make them lose valuable work, but in DSDM, this is iterative in small increments, hence the loss if any is small [28].
- **Base-line the requirements:** Some high-level requirements must be set early in the process to limit the amount by which the requirements can be changed. These basic requirements, often thought of as a "freeze" on changes, occur during the business study phase of the process [28].
- **Testing throughout the lifecycle:** Most of the development techniques in existence demand testing at the design or implementation phase, as late as that. According to DSDM, testing is done right at the beginning of the development process. Even testing interview documents by cross checking, them with a control group or similar techniques [28].
- **Stakeholder collaborative approach:** It becomes vital to have an atmosphere of trust and honesty in order to capture precise requirements and honest feedback on a resultant product. In DSDM, collaboration between Business staff and developers is of utmost importance in delivering value. Precise business needs along with honest feedback helps in quick development which thereby results in timely delivery of the project [29].

The DSDM approach typically has 6 phases, consisting of an incremental and iterative design that prioritizes critical business requirements early in the process. This approach represented in figure 5 allows business stakeholders to see and feel the emerging solution and give their feedback and requests to change along the path of development [30]:

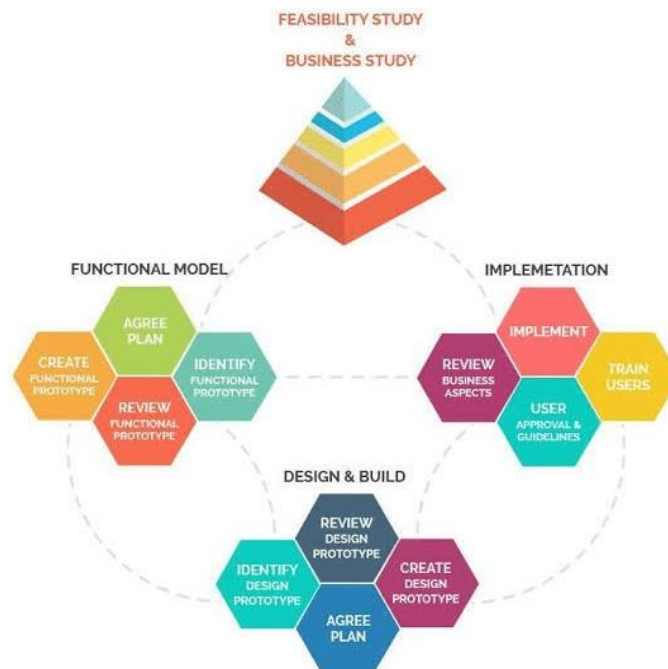


Figure 5 - DSDM lifecycle phases (Source: [29])

- **Pre-Project Phase:** Consistent with the DSDM philosophy, which states “that best business value emerges when projects are aligned to clear business goals”, [30] the Pre-Project is set up in such a way that projects are prepared on an objective-set basis, so we need to make sure resources exist to start the project's feasibility stage.
- **Feasibility Phase:** Check the project's specifications to identify whether the goals can be accomplished with the available budget, timeline, and technology. If the plan looks unrealistic, edit it so that your plan is aligned with your team's capability sets [31].
- **Business Study Phase:** Once technical feasibility is cleared, a review of the project from a business perspective is needed to check that it aligns with the strategic goals of the organization and stands competitively in the industry to meet user needs [31].

- **Functional Model Iteration:** In this phase, from earlier specifications, we will develop and continuously test a prototype with a guarantee it works and has all the critical features included. Refine the prototype until it accurately reflects what the final product of the desired outcome is [31].
- **Design and Build iteration Phase:** Now the focus is to iterate on the prototype until it is as efficient as possible and implement more pieces to reinforce the project and fix any problems. Properly test the product to make sure it is responsive, fully functioning and easy to understand [31].
- **Implementation/Deployment Phase:** The Deployment phase is the aim to put a baseline of the Evolving Solution into operational usage. The release deployed by the Deployment Phase may be the final solution or a subset of the final solution [31].

Understanding the DSDM requires a balanced view of its strengths, against its limitations and this analysis is reflected in table 4.

Table 4 - Analysis of the Strengths and Weaknesses of the DSDM Model

Strengths	Weaknesses
- Improves collaboration and supports rapid deliverables by promoting communication between cross-function teams. - Increases project organization, since teams apply specific tools and strategies to streamline development and keep on schedule - Sets clear guidelines: Continuous engagement of the customer throughout the project ensures alignment with client expectations and business objectives [31].	- The High initial cost involved due to the need to train team members, acquire proper tools and dedicate resources. - The model requires constant oversight and active participation from project managers and stakeholders, which can be intensive and limit creativity. - Due to its comprehensive structure, DSDM can be overly complex and unnecessarily detailed for smaller or less complicated projects [31].

2.2.5. Rational Unified Process (RUP)

The Rational unified process (RUP) is an object-oriented system development methodology originally offered by Rational Software, which is now part of IBM.

The RUP was developed by Grady Booch, James Rumbaugh, and Ivar Jacobson, the three pioneers behind the success of the Unified Modelling Language (UML) [32] and it uses not only UML's modelling techniques but also guidelines, templates and tools to ensure effective and successful development of software systems [33]. The tools offered by RUP allow a large part of the development process to be automated, including modelling, programming, testing, managing the project and managing change. RUP spread rapidly, with 10,000 companies using it as of 2003, the year Rational was acquired by IBM, it was one of the earliest iterative methods to be widely publicized and was defined a half decade or more before the agile manifesto methods began to appear [34].

A phase in the RUP can be thought of as a goal or major emphasis for a particular portion of the project. The four phases of the RUP life cycle depicted in figure 6 are Inception, Elaboration, Construction and Transition:

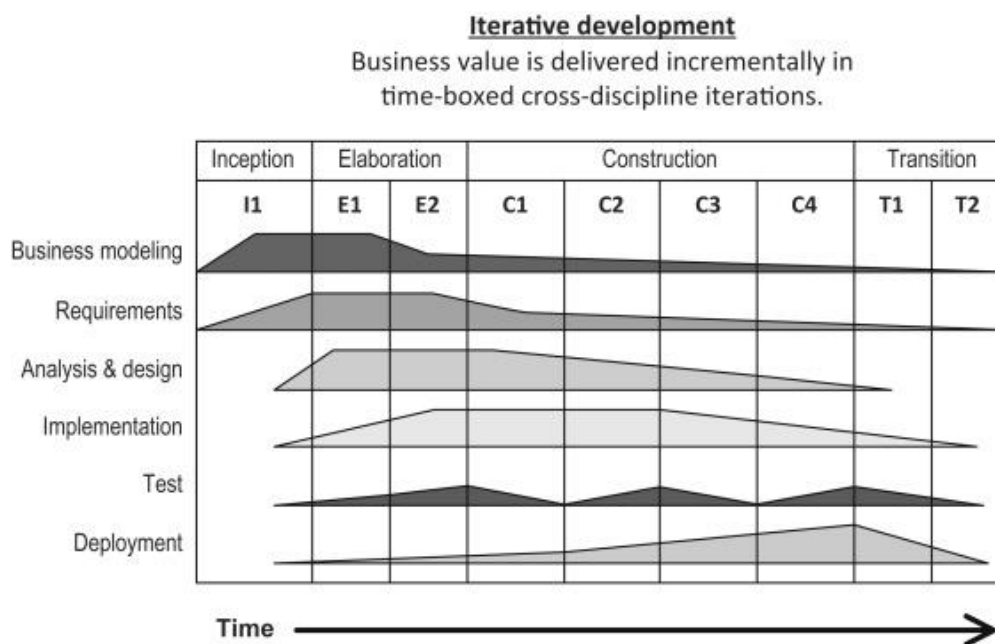


Figure 6 - Rational Unified Process Whale Chart (Source: [34])

- **Inception:** The major objectives of the inception phase are to identify the business scope of the new system and the project itself, develop initial cost and schedule estimates by mutual agreement of the stakeholders, acknowledge the business rationale for the project, rationalize the requirements in light of the business case for the project, and define a vision for the proposed solution [35].

- **Elaboration:** This phase involves the collection of detailed information on what will establish both functional and non-functional requirements of the new system. Functional requirements describe the tasks the system should perform, and non-functional requirements discuss technical specifications and usability, security, and other related elements. The requirements are prioritized and reviewed in various structured walkthroughs with users to ensure accuracy and appropriateness. Typically, at this point, user interface designs are created as well [35].
- **Construction:** In this stage, more emphasis is given to coding and testing. All the parts of the system, including user interfaces, business logic, and data access functions, are developed according to earlier design specifications. This phase aims to deliver a working system ready for the next phase of deployment. In fact, the construction phase may involve several iterations, especially in big projects, to control the workload and fine-tune the system components [35].
- **Transition:** During the transition phase the system is deployed to production and made available to the end users. The processes of this phase may include iterations of training sessions with user manuals, beta testing for user expectation matches, and any necessary adjustments and optimization and upon completion of these requirements, the development cycle is considered complete [35]. After the system is in operation, it will need to be supported and maintained [34].

In order to fully understand the Rational Unified Process, it's important to assess both its pros and cons. Table 5 details the primary strengths and weaknesses of the RUP model, hence assisting in analysing it for any software development project. This comparative overview will give a clear view on where the RUP excels and where it is lacking.

Table 5 - Analysis of the Strengths and Weaknesses of the RUP Model

Strengths	Weaknesses
- RUP ensures detailed recording of each phase, which is invaluable for projects that involve multiple collaborators. - RUP equips software teams to address and mitigate potential issues early, enhancing their ability to manage risks and resolve problems effectively. - A fundamental component of the RUP methodology is the consistent provision of updates to all project stakeholders, ranging from team members to executive leadership and external vendors. - RUP can help reduce the duration required for development and integration phases, streamlining the overall project timeline [36].	- RUP is a fairly involved process, requiring a great deal of experience on the part of team members working with software. If your team includes beginners, you may want to consider a less complex development process. - RUP requires a lot of documentation, a very expensive process in both monetary and time capacity. Teams with scarce resources will want to consider more affordable means of development. - On larger projects with many components supported by multiple software development teams, the RUP could add more complexity during its integration. - The focus on continuous integration may demand slowing down or considering other methodologies better suited to multitasking across multiple development streams [36].

2.2.6. Agile Methodology

Following the discussion from both traditional and modern software development methodologies, the focus now turns to Agile. Every technology organization these days seems to practice some version of agile methodology, but as demonstrated before that was not always the trend [37].

According to Highsmith: “Projects may have a relatively clear mission, but the specific requirements can be volatile and evolving as customers and development teams alike explore the unknown” [38]. This highlighted issue, along with others revealed many of the weaknesses of traditional and recently called modern software methodologies.

In 2001, seventeen professionals came up with the manifesto for agile software development [39]. They wanted to outline values and basic principles for better software development. Besides being brought into focus, the manifesto has seen extensive use by developers throughout software-developing organizations and beyond the world of IT. Principles of Agile and their implementation in practice have opened the way for new and innovative approaches to software and product development [40]. While not always all the founders were always in agreement, they did find consensus around four core values and twelve principles on their vision on how an agile world should be defined:

Agile Manifesto Core Values [39]:

- Individuals and interactions over processes and tools;
- Working software over comprehensive documentation;
- Customer collaboration over contract negotiation;
- Responding to change over following a plan;

Agile Manifesto Principles [39]:

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software;
- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage;
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale;
- Businesspeople and developers must work together daily throughout the project;
- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done;
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation;
- Working software is the primary measure of progress;

- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely;
- Continuous attention to technical excellence and good design enhances agility;
- Simplicity—the art of maximizing the amount of work not done—is essential;
- The best architectures, requirements, and designs emerge from self-organizing teams;
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behaviour accordingly;

Clearly, not every agile process model may uniformly apply the 12 principles in equal measure, and some models may downplay or even completely ignore certain principles altogether. The variability is, however, there to permit its application according to the project needs and contexts [41].

Even though the processes and practices vary for each of the different Agile frameworks, all projects follow the same Agile lifecycle. The Agile software development lifecycle defines the sequential phases a project follows from its beginning to its eventual delivery.

This lifecycle is divided into six steps: **concept, inception, iteration, release, maintenance, and retirement**. While methodologies can differ, this consistent framework ensures that the progression through these phases offers a structured approach to manage changes and deliver value efficiently throughout the project. This structure available in figure 7 not only enhances adaptability but also ensures that projects stay aligned with evolving business objectives and stakeholder needs [42].

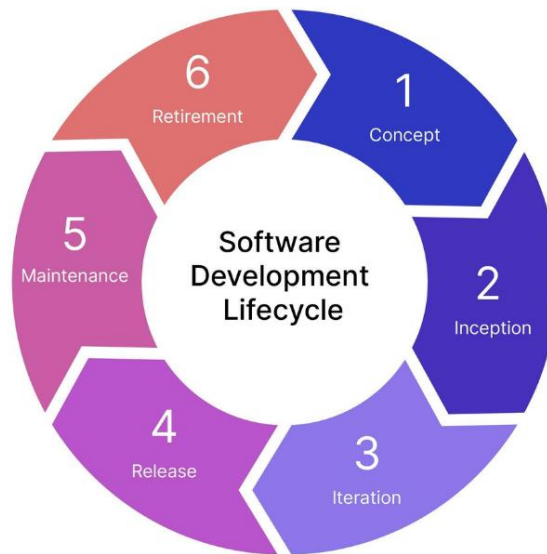


Figure 7 - Agile Software Development Lifecycle (Source: [42])

- **Concept:** This is the first step of the lifecycle. In this stage, the product owner defines the scope and prioritizes critical or urgent projects. After this, the product owner consults with stakeholders regarding suitable key prerequisites and begins laying out the project by focusing first on necessary features and desired results. It is advisable to keep project requirements minimal at this stage, allowing for later adjustments in the lifecycle. Estimates of costs and timelines are also taken care of in this phase, and any resource or time constraints are assessed to decide whether to proceed. For projects introducing new solutions, a Proof of Concept will be developed to assess technical viability [42].
- **Inception:** Immediately after the concept phase is the inception phase, where the software development team is formed according to the project's needs. In addition to selecting individuals with the right skill sets, the product owner ensures that the team has the necessary resources to succeed. This is when the designers also work with the developers in designing the user interface and setting the architectural basis [42].

- **Iteration:** Iteration or Construction is one of the important aspects of Agile and tends to be the most time-consuming since much building and testing happens at this stage. Most iteration phases are divided into sprints, which are short work cycles, typically two to four weeks in duration, with each sprint intended to deliver functional software. It starts with focus on core features, and while a project develops further, user feedback is integrated into the code along with improvements [42].
- **Release:** The Release/Deployment phase of Agile's lifecycle itself needs to be the actual execution of planned incremental releases in accordance with the prioritized backlog. The phase has to deal with introducing new functionalities while keeping the whole system intact by using feature toggles. This will involve writing automated deployment scripts, which deploy consistency, coupled with an automated post-deployment smoke test that confirms stability of the system [43].
- **Maintenance:** In the maintenance stage of Agile software development, continuous monitoring of the software should be done to make sure that at each instance it is effective to the users and relevant also. This stage mainly involves the continuous monitoring of the software in order for defects and issues arising during use to be identified and fixed. The feedback will be retrieved from several sources and is then used to identify and prioritize problems that need attention. Other than immediate fixes post-release, maintenance in Agile is meant to extend the life of the product with relevance to users [44].
- **Retirement:** Product retirement by Agile transitions the users to a new software product so that they do not experience user drop-off. The process begins with informing users of the discontinuation of the current product and the implications that comes with that effect, assuring them of a smooth transition into a newer system and the transfer of their data there. Normally, retiring consists of developers retiring the old product by slowing down its support and updates before discontinuing it altogether [44].

With the emergence of internet applications, software development began to evolve. Much of the early development was done in startup environments: with smaller, differently allocated teams, often without strict computer science backgrounds. Above all, there was great financial and competitive pressure to rapidly deploy websites, apps, and new features. In response, development tools and platforms quickly started to change [37]. This methodology was a “game-changer” introducing a more flexible and iterative approach to software development, emphasizing rapid response to change and close collaboration with stakeholders.

2.3. The Growth of Low-Code Technologies

Low-code development platforms emerged in the early 2000s as a response to the increasing demand for faster software delivery, but it was only in 2014 that the term “Low-Code” was introduced by Clay Richardson and John Rymer in a Forrester research article [45], where they stated that “Some firms are now turning to new, “Low-Code” application platforms that accelerate app delivery by dramatically reducing the amount of hand-coding required and the usage of low-code platforms is now gaining momentum for customer-facing applications”.

The authors noticed that the number of firms leveraging low-code platforms to rapidly develop new programs was increasing. First, firms that are using these internally and generally were impressed by the speed at which such platforms could create new applications. Soon, they began to perceive how effective those tools would be for customer-facing products, too, by making use of the infrastructure and reusable components of the platforms [46]. This led to a vast growth in the customer base and valuation of such notable platforms provided by established companies like Appian, OutSystems, and Mendix. Over the past few-years these Low-Code companies also experienced a significant growth in their market value, for example current valuation of OutSystems stands at 9.5 billion US dollars [47], while Mendix was acquired by Siemens for 700 million dollars back in 2018 and also, US-based researches and the consulting firm Gartner forecasted that the global Low-Code market revenue will reach \$31.9 billion by end-2024 [47].

Some key factors have fuelled the growth of low-code technologies, and filled gaps previously left empty by other traditional methods, such as their ability to reduce or eliminate the need for manual coding in software application development and deployment, the downsize in the associated time and cost investments to their lowest possible levels and simplification of training required for teams tasked with creating, maintaining, or evolving these applications [2].

With the aim to perceive how the IT community was reacting to this growth of LCNC technologies OutSystems carried out a study in 2019 labelled as “The State of Application Development: Is IT Ready for Disruption?” [48], that presented the results of a survey answered by more than 3300 IT professionals across different continents with different roles (Application Developers, IT Leader/CIO/CTO, Business Analysts, Data Specialists and others) in different companies with varied dimensions so that these same companies could encompass all types of industries (Software, Government and Education, Insurance, utilities, retail and consumer and others).

The primary motivations for adopting low-code development platforms, were interpreted and depicted in the radial charts that are available in figure 8 and figure 9 [49]:

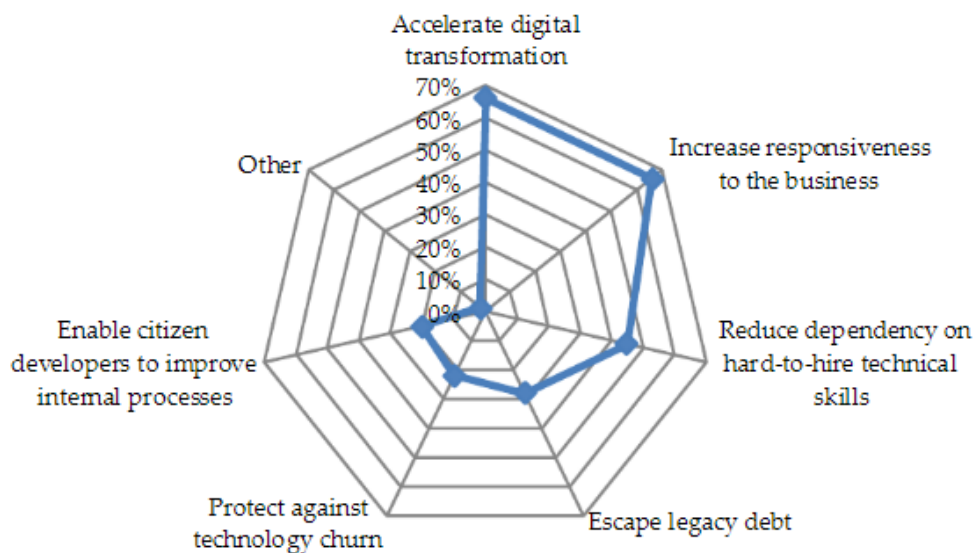


Figure 8 - Main reasons for using LCDP according to IT Users (Source: [49])

The highlights of the main benefits of Low-Code development platforms:

- **Velocity:** Since the major part of the code is already developed, all users have to do is visually configure the apps, instead of hand-coding, or make necessary adjustments to develop the apps that they need [48].
- **Cost Reduction:** Once the development cycle is reduced due to a time viewpoint, the cost is also reduced, whether the app is developed by the company or by external developers [45].

- **Ease of Complexity:** Since the apps are not from scratch, the apps development is simplified, and this fact enables the ability to pay more attention to tuning the software to fulfil users' needs [45].
- **Low maintenance:** Justified because maintenance is one of the most fundamental stages in software development, which helps deploy changes fast in existing features, in order to fit a business demand that usually changes very fast. Low-code development platforms, by their very nature set up for minimum coding, require less code to be maintained, hence streamlining the update process [49].
- **Involvement of business profiles:** These platforms provide very usable and smooth interfaces for app deployment. In this regard, technological knowledge is not required, whereas end-users with extensive business knowledge play and act as app developers. This is evident in the fact that 44% of the users on low-code development platforms are business professionals who work alongside IT teams [49].

Along the research, the main key factors for avoiding or not considering low-code development platforms as well as its downsides appointed by users were also interpreted and charted in figure 9.

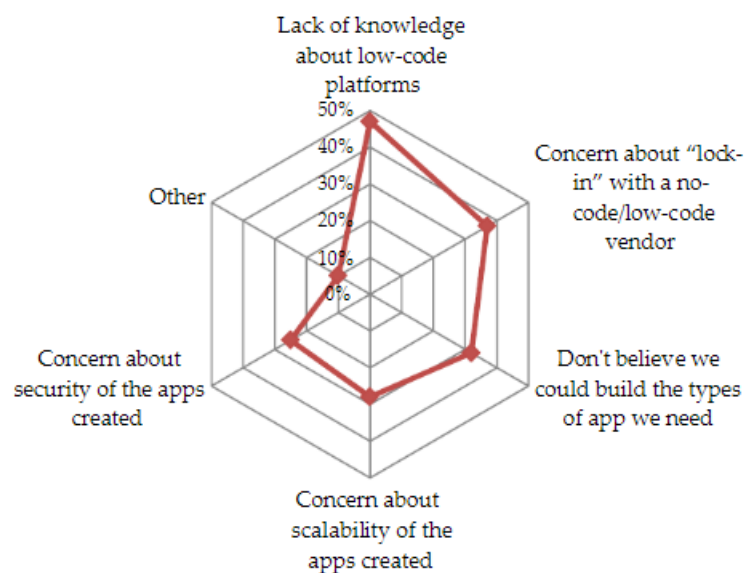


Figure 9 - Main reasons for not using or considering low-code development platforms (Source: [49])

It is also important to notice that most of these concerns were raised a few years ago, based on the current knowledge of IT users and as of recently, many low-code platforms have addressed these issues through several enhancements. However, according to the study analysis conducted by OutSystems, these are the key points of the principal limitations and concerns associated with adopting low-code development platforms:

- **Scalability Concerns:** According to the authors, Low-Code development is designed for simpler applications, and thus they can't handle huge data flow processing at once. Despite being an issue that was already addressed by most of the important Low-Code “big players”, the authors believe that applications built by a citizen developer should be for small digital needs and not suitable for complex tasks [50].
- **Fragmentation:** One recognizable problem within Low-Code development is fragmentation/, drawing upon different paradigms according to the particular vendor and his own peculiarities in the programming model. Thus, multiplicity across various platforms may become a challenge because that would mean every different vendor could introduce his own set of rules, and even his own set of tools, which actually impairs developers from being able to switch between the different low-code systems or integrate these as such [49].
- **Software-Only Systems:** Even though enterprise developers may not be deeply experienced as programmers, they usually are experts in other branches of engineering. These practitioners expect to apply their domain-specific knowledge in programs, which must be written at an appropriate abstraction level using common formalisms within that domain. However, with too narrow focus on software at the expense of accommodating other engineering disciplines, this can lead to reduced utilization of expert knowledge spanning areas and undermining broader applicability and usefulness of solutions they can develop [50].

2.4. Low-Code Platforms Use-Cases

If we imagine a scenario where in a business-critical moment a company needs rapid development and that is the core of a meeting focused on high-priority organizational needs, we must realize that traditional development cycles face severe bottlenecks, hence taking much longer and delaying strategic initiatives and in today's fast-moving business world, waiting several months for deployment is unsupportable [51].

This is where the low-code platforms come into play. It allows for rapid application development without compromising on the quality of it. To understand its reach and adaptability, it's necessary to delve deep into its most common use cases and understand where this technology comes into play with its true potential.

The use cases for Low-Code development platforms span from very simple to the most complex or mission-critical applications and touch almost every area of enterprise application development:

- **B2C Mobile Apps:** Building and maintaining a B2C mobile application no longer necessitates a large team of software developers. Thanks to low-code platforms, businesses can now create cross-platform mobile apps that deliver robust functionality to users. This streamlined approach not only reduces development time but also allows companies to respond more quickly to evolving customer needs and market trends [52].
- **Customer portal Apps:** Nowadays customers want fast, self-service access to their accounts and services with no need to contact support. Customer portals offer them this frictionless experience where customers can interact with businesses whenever and wherever. Low-Code platforms make it faster and easier for organizations to build fully tailored, user-friendly portals meeting the needs of both business and customer alike in but a fraction of the time [51].
- **Employee Apps:** Employees want to use tools that multiply their efficiency, instead of tools that would slow them down with outdated systems. In its place, Low-Code development enables the organization for rapid creation of intuitive, personalized apps that remove work complexity and multiply productivity. Whether it is automating mundane tasks, making data entry simpler, or presenting real-time insights, Low-Code makes application creation really simple-easy, in fact-that employees will actually use and enjoy [51].
- **Customer Relationship Management (CRM) and Automated Sales Processes:** Low-code platforms enable businesses to enhance customer engagement and boost sales efficiency by optimizing sales processes and building or improving CRM systems [51].

- **Supply Chain Management:** Customers want to order goods through their mobiles, but many business supply chains are dependent on spreadsheet-based and, worse still, entirely manual obsolete systems. Mobile-based supply chain portals smoothen the field operations. Institute real-time inventory tracking, use automation for low stock or shipment delay alerts, and efficiently create purchase orders and invoices by using a low-code platform [52].
- **IoT Integration:** Low-code applications can seamlessly integrate with IoT platforms, enterprise systems, and third-party services, allowing organizations to turn data into actionable insights for users. These development platforms can help businesses create engaging customer experiences from compatible/connected devices, such as medication temperature monitors, equipment trackers, service issue warnings, among other IoT use cases [53].
- **Next Generation Technologies Apps:** Many Low-Code platforms provide a very comprehensive library of components developed by top technology companies. They also facilitate seamless integrations with third-party services hosted on the cloud, such as AI, machine learning, and blockchain. This allows businesses to enhance their apps with advanced capabilities, driving greater efficiency and allowing innovation without the complexity of hard coding [53].

Low-Code Application Platforms Analysis

Chapter Introduction

This chapter will introduce the most influential low-code platforms to date, those that reshape the software development landscape with respect to its architectures and core components.

As businesses increasingly adopt low-code platforms to drive faster delivery of applications, the likes of OutSystems and Mendix create leadership momentum by providing full-fledged tools and infrastructures that support enterprise-grade development.

In this chapter will focus on a review of OutSystems, a fully cloud-native platform that comes with an architecture and a set of tooling for application lifecycle management, having transformed app development. Next, the focus turns to Mendix, covering its solid architecture, the main components in depth, and how it can integrate with various enterprise systems. This chapter is based on an informative and practical analysis of such platforms, given that the low-code technologies have accelerated businesses by making digital transformation easier through innovation and agility.

3.1. Leading Low-Code Platforms

The rise and adoption of low-code platforms has become a trend between companies and is rapidly reshaping the software development landscape, offering businesses a faster, more efficient way to build applications. According to Gartner, by 2023, more than 50% of medium to large enterprises have adopted a Low-Code Application Platform (LCAP) as a strategic component of their application development strategy [54].

Gartner defines low-code application platforms (LCAPs) as tools designed to enhance the development and deployment of custom applications by reducing the need for extensive programming and enterprise LCAPs as their subset targeted at a wider range of possible developers, especially in large organizations, including critical features to support the creation, delivery, and maintenance of applications in middle-sized and large enterprises [55].

In 2023 Gartner launched its most recent and well-known Magic Quadrant analysis, depicted in figure 10 that evaluates the most distinguished Enterprise Low-Code Application Platforms:

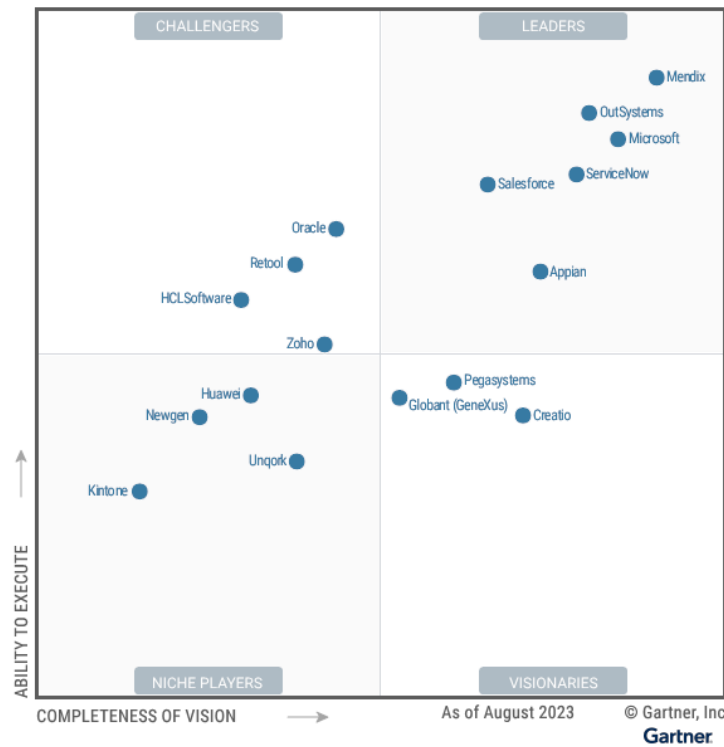


Figure 10 - Gartner's Magic Quadrant for Enterprise Low-Code Application Platforms (Source: [55])

This type of evaluation aims to position technology players within a specific market (Low-Code in this particular case), by applying a graphical treatment and a set of uniform evaluation criteria as it is illustrated in figure 11, a Magic Quadrant enables you to quickly determine how well technology providers are executing their stated visions and how well they are performing against Gartner's market view [56].

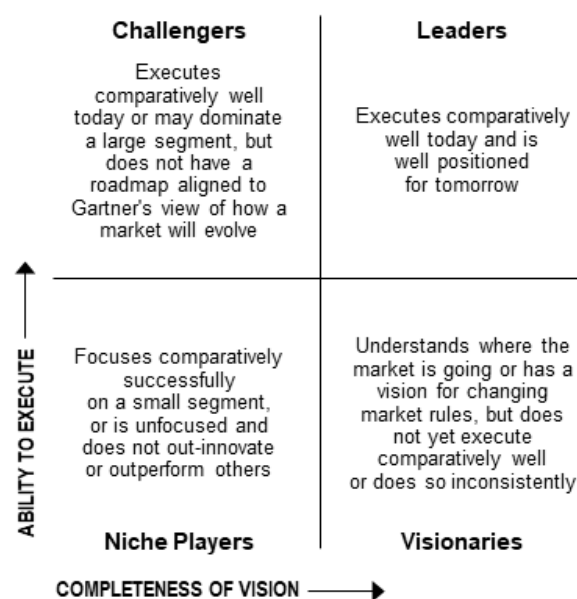


Figure 11 - Gartner's Magic Quadrant structure explained (Source: [56])

As displayed in figure 11 it is possible to verify that the foundation of the report is to evaluate low-code platforms taking two major dimensions into account: 'Completeness of Vision' and 'Ability to Execute.' With these dimensions being themselves constituted of several factors, each weighing differently in the process:

- **Completeness of Vision:** Gartner gives broad importance to innovation, market understanding, and product strategy. Besides, the business model, degree of innovation, and marketing and sales strategy will also be taken into consideration [57].
- **Ability to Execute:** In this context, the focus is mainly with the product capability of the platform and the efficiency in sales and pricing strategies. The other key concerns are overall viability, customer satisfaction, and the responsiveness of the platform to changes in the market [57].

In the last edition of Gartner's Magic Quadrant for LCAP it is possible to identify six leaders: Mendix, OutSystems, Microsoft PowerApps, Salesforce, ServiceNow, and Appian. However, it is important to consider the strong market position that both Mendix and OutSystems have maintained over the years. As shown in **Appendix A**, they remain as indisputable leaders in this sector.

In 18 of September of 2023 Gartner launched one of its most recent researches on LCAPs [58], where several use-cases were considered in order to evaluate and compare different LCAPs products to verify whether they would solve specific use-cases, where one of which strongly aligns with the goal of this research "Composing applications for Business-2-Employee (B2E) departmental and enterprise applications".

The products were evaluated according to a set of critical capabilities, but most importantly these LCAPs need to follow also some mandatory business criteria that alone already represent a strong indicator of why these LCAPs are top contenders for market leadership:

- **LCAP should have a license or subscription revenue of at least \$60 million** over the previous year, and a minimum of 100 paying enterprise customer organizations (of at least 1000 employees) for its LCAP offering, excluding other related product offerings [58];
- **LCAP should have a license or subscription revenue of at least \$25 million** over the previous year, and a minimum of 5000 paying enterprise customer organizations for its LCAP offering, excluding other related product offerings [58];

- **LCAP developer community** of more than 100k developers using the LCAP across all customers [58];
- **A significant market relevance index** (Customer Interest indicator [CII]) of a minimum score of 50% [58];
- **International presence** within 3 or more of the following geographies (Europe, North America, South America, Middle East and Africa, China, Japan/Asia/Pacific) [58];

As displayed in **Appendix B**, both Mendix and OutSystems received one of its highest scores, for this use-case (4.73 and 4.86 specifically), in a scale where 1 is equivalent to “Poor or absent: most capabilities not achieved” and a 5 is equivalent to “Outstanding: significantly exceeds requirements”.

So given these evaluations, it is now relevant to delve into a more detailed examination of these two leading platforms and explore their capabilities further.

3.2. OutSystems

Founded in 2001, with a vision to transform enterprise software delivery. OutSystems has grown to become one of the largest Portuguese tech unicorns, when in 2018 secured a \$360 million investment from KKR Investment firm and Goldman Sachs [59]. The Low-Code platform helps organizations in the fast development, deployment, and management of custom applications with minimal coding. It offers a visual development environment wherein users can build applications by assembling pre-designed components via a drag-and-drop interface. In this way, there is little reliance on extensive manual coding; hence, this platform is open to both seasoned developers and nontechnical users, who are sometimes referred to as 'citizen developers' with limited programming skills. Whether the business operations are on-premises, in a private cloud, or in a public cloud like (AWS, Azure, Google Cloud, etc.) OutSystems supports seamless integrations [60], enabling the selection of a deployment option that best aligns with an enterprise's infrastructure:

- **Self-managed IaaS:** OutSystems 11;
- **Cloud-ready PaaS:** OutSystems 11 Cloud;
- **Cloud-native:** OutSystems Developer Cloud (ODC);

3.2.1. OutSystems Cloud and OutSystems 11 Platform Architecture

The architecture is the same both for the OutSystems PaaS cloud-ready solution (OutSystems Cloud), and for the self-managed IaaS option (OutSystems 11). It includes all the components, tools, environments, and code generation needed for developing extensible applications that run on standard .NET servers. With OutSystems, there are no tie-ups to proprietary technologies, giving enterprises the flexibility to move freely across environments. Furthermore, this architecture makes enterprise-grade app development highly scalable [60].

To effectively monitor and oversee an application's development from its initial concept through design, development, testing, deployment, and continuous maintenance, OutSystems has environments, tools and components that cover the full application lifecycle management process as depicted in the figure 12:

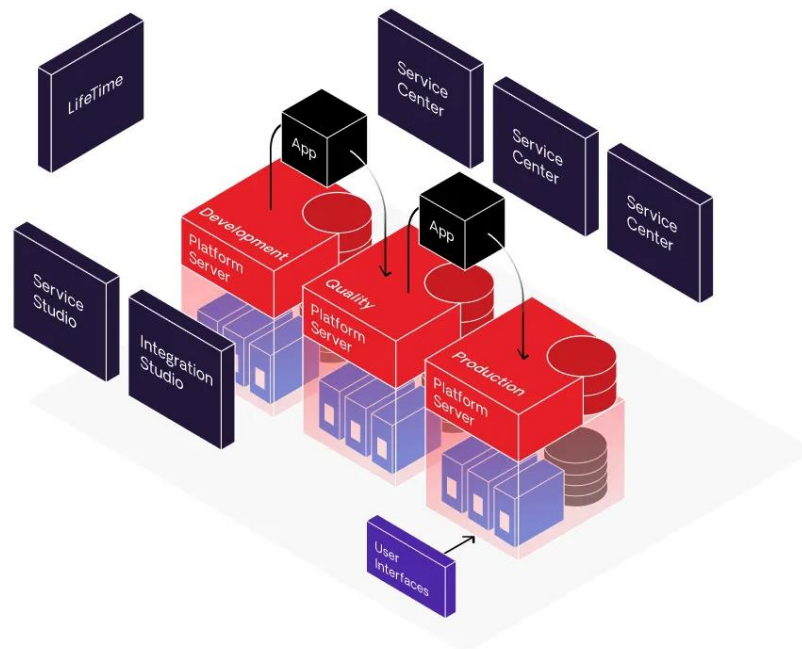


Figure 12 - OutSystems application lifecycle management components (Source: [60])

Tools and Components:

- **Platform Server:** The Platform Server is the core element that orchestrates all runtime, deployment, and management tasks across applications. It must be installed in each environment to ensure the proper functioning of the system. This system ensures application performance and consistency across the various stages of the development lifecycle [61].

- **Service Studio:** Service Studio illustrated in figure 13 is the environment within which all the various aspects of an application stack are built in OutSystems from the data model, application logic to user interface, business process workflows, integrations, and security policies. Service Studio is the core development environment for the platform, providing a visual interface to create, test, and deploy applications in a low-code manner [60].

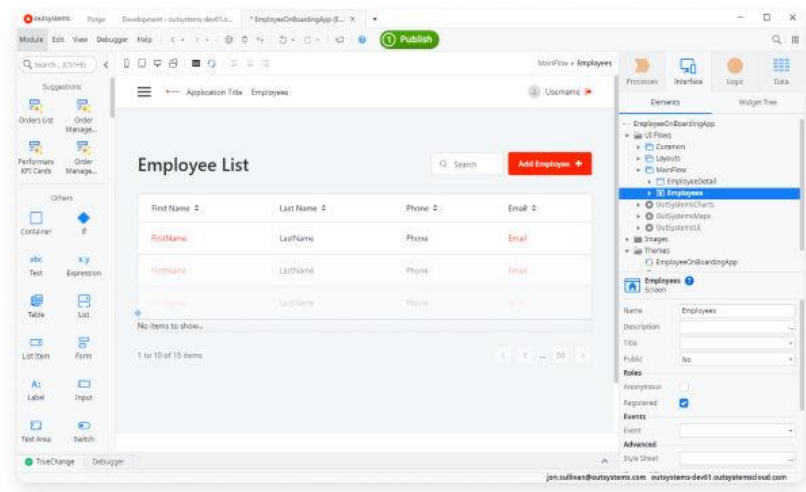


Figure 13 - OutSystems Service Studio common use case (Source: [62])

- **Integration Studio:** This is the environment used to develop components that extend the functionalities of the OutSystems platform. Represented in figure 14 it allows the inclusion of code written in external systems and third-party integrations. Integration Studio is a dedicated tool in OutSystems that enables developers to integrate external databases, web services, and custom code into the platform, ensuring seamless connectivity and extended functionalities [61].

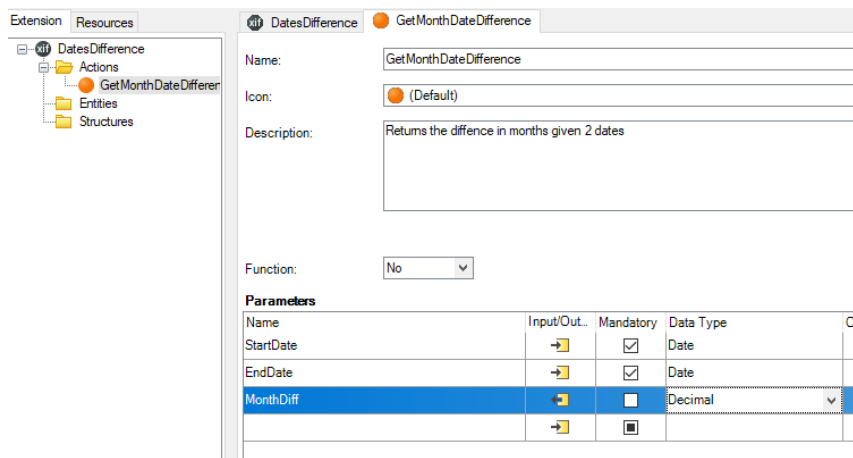


Figure 14 - OutSystems Integration Studio .NET extension use case (Source: [62])

- **Builders:** Builders are SaaS (Software as a Services) products intended to automate and expedite manual, repetitive tasks such as the development of workflow apps, enhancement of UX/UI to provide quality mobile experiences, and the integration of the system with other external systems. Builders speed things up by providing ready-made components and templates that save developers time, allowing them to focus on innovation rather than repetitive tasks. In this respect, they are valuable tools for reducing time and effort while delivering scalable, enterprise-grade applications [60].

Management Consoles:

- **Service Center:** The service center handles the various operational settings of an environment, from connection settings to web service endpoints, and application properties. It provides a unified console for administrators to monitor the system, manage deployment, and perform maintenance tasks, ensuring stability and performance across applications within the environment [60].
- **Lifetime:** LifeTime management console is the centralized console for managing your OutSystems environments, applications, IT users, and security, covering the full application life cycle from development to deployment [61].

On the following image 15 and 16 it is possible to identify the environments of a typical OutSystems infrastructure that can be used both for OutSystems Cloud or OutSystems 11 (Development, Quality and Production), which follow the same order whether the platform is running on a private-cloud or on-premises:

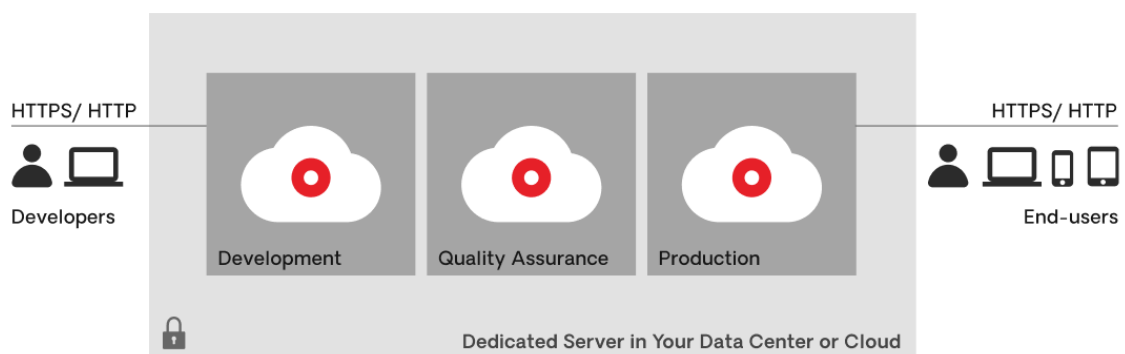


Figure 15 - OutSystems running on a private cloud (Source: [61])

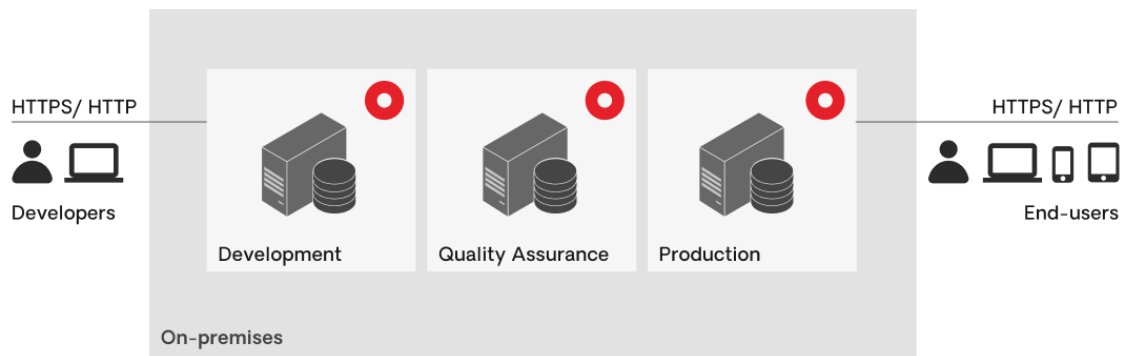


Figure 16 - OutSystems running on-premises (Source: [61])

3.2.2. ODC Cloud-Native Architecture

According to the research “Cloud-Native Development Report: The High Cost of Ownership” [63] the total cost of ownership (TCO) of a traditionally developed cloud-native approach is, on average, \$5.6 million and can take up to 18 months. This is where OutSystems’ newest solution, **ODC** (OutSystems Developer Cloud), plays a crucial role, offering a cloud-native platform that significantly reduces both development time and costs by providing a modular, scalable environment built on best practices in cloud-native infrastructure and operations [63].

ODC is a cloud-native application development platform designed to meet the most critical and strategic needs of businesses. It offers a modular, scalable and flexible environment for developing, deploying and managing applications. Built with modern cloud-native architecture principles, it incorporates best practices in infrastructure management, operations, and security. This ensures innumerous integration with cloud environments, enabling organizations to leverage cloud scalability and flexibility while maintaining high performance and operational efficiency [60].

Additionally, ODC simplifies development by providing out-of-the-box modules, automation tools, and continuous integration/continuous deployment (CI/CD) capabilities, allowing businesses to accelerate innovation while maintaining strong governance and control.

By deeply analysing the ODC architecture diagram displayed in **Appendix C**, it is possible to breakdown the components displayed into the following 7 groups [63]:

Security and Access Management

- Web Application Firewall (WAF)
- Content Delivery Networks (CDN)
- Identity Service

Development Environment

- ODC Portal
- ODC Studio

Traffic Management

- Load Balancer

App runtime Management

- Kubernetes Cluster
- Runtime Environments

Data Management

- Application Data
- Key Management
- Secret Management

Messaging and Communication

- NATS

Monitoring and Analysis

- OutSystems Analytics Pipeline

In the ODC architecture composed by these enumerated elements and displayed in **Appendix C**, the users get access to applications through a **CDN** (that distributes the app content globally), secured with a **WAF**, which filters and protects against malicious traffic. The users get to interact with the system via the **ODC Studio** which is the environment where developers create and manage applications and use the **ODC Portal** to manage deployments and configurations.

Once users are authenticated by the **Identity Service**, requests are routed by the **Load Balancer** to the **Kubernetes Cluster**, where applications like App 1 and App 2 run in isolated containers. These applications scale dynamically within the cluster, managed by **Kubernetes**.

NATS (which is a messaging system that facilitates communication between distributed services or microservices within the platform) handles asynchronous messaging between services, ensuring efficient communication. **Application data** is stored securely, with **Key Management** encrypting sensitive data and **Secret Management** safeguarding credentials like API keys.

Finally, the **OutSystems Analytics Pipeline** collects and processes performance metrics, offering real-time insights into application health and efficiency. This setup ensures that interactions are secure, scalable, and optimized for both development and production environments.

3.3. Mendix

Mendix is an industry-leading low-code application development platform, founded in Netherlands in 2005 by three founders who understood applications in general weren't failing for technical reasons, the problem was the communication. Developers didn't understand the business, and business users didn't understand development. In 2014 Battery Ventures (an American global investment firm) invested in Mendix, supporting their expansion into the US market [64].

Few years after this investment, the fast-growing tech company doubled its sale resources and started to have a much stronger position between larger clients and enterprises, and in 2018 it was acquired for \$730 million by the giant Siemens, since then, it never stopped growing.

Mendix offers an entire set of portfolio management, Agile planning, AI-driven intuitive development, and built-in version control tools, all integrated from idea to development, deployment, and maintenance [65].

Mendix leads in the development of enterprise applications [55] because it offers an encompassing progress through seamless integration of data from other systems, automation of processes, and a simplified manner of deploying and managing cloud applications. But how does Mendix accelerate speed and quality by enabling collaborative low-code development? The answer to this question can be seen in figure 17.

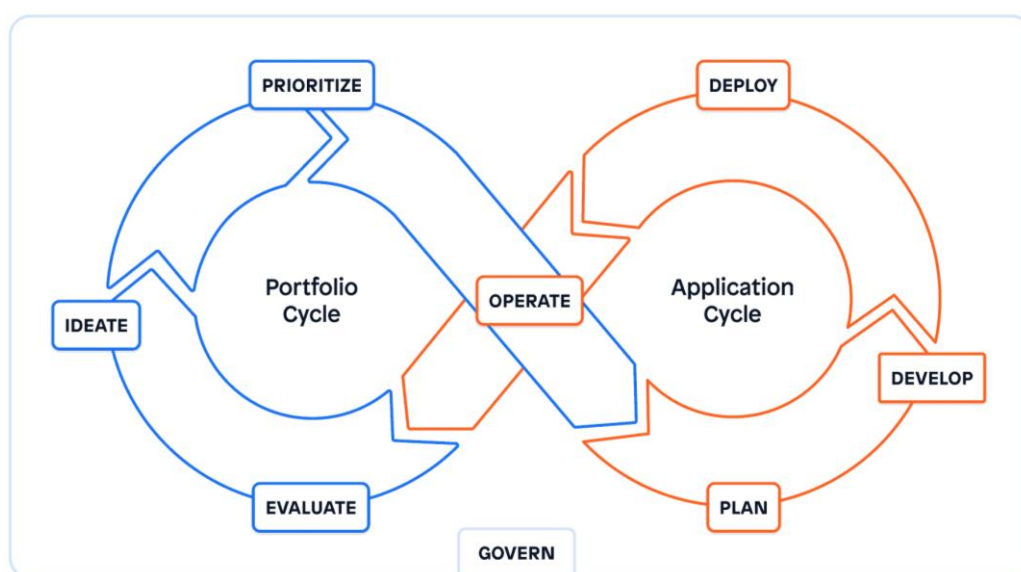


Figure 17 - Mendix Software Development Life Cycle representation (Source: [65])

In figure 17 it is represented the Mendix development lifecycle, which provides a structured approach to manage the end-to-end process of application development.

By dividing the lifecycle into two distinct but connected cycles (platform management and Application lifecycle) Mendix assures a continuous innovation and operational efficiency and if used well, this model will ensure companies are able to prioritize their app portfolios effectively and govern them while being able to develop, deploy, and maintain applications with a high degree of productivity [65].

Portfolio Cycle: (This cycle represents the higher-level planning and prioritization activities for multiple projects and applications. Consisting in 4 important stages, according to the Portfolio Cycle, organizations consistently match application development with strategic objectives) :

- Ideate
- Evaluate
- Prioritize
- Operate (Shared stage)

Application Cycle: (This cycle focuses on the development of individual applications. Consisting of 4 different stages) :

- Plan
- Develop
- Deploy
- Operate (Shared stage)

Across both cycles, **Governance** makes sure that the entire process complies with security, compliance, and other organizational standards in both cycles.

When using the Mendix platform, by following this lifecycle businesses can effectively develop and maintain individual apps while managing their application portfolio [65].

Mendix Platform Architecture

The Mendix Platform is an integrated aPaaS (application Platform as a Service) solution which aims to design, build, deploy, and manage enterprise applications. It provides developers and administrators with tools and services through the Mendix Portal, from requirements management up to deployment.

The platform includes various distinct components, displayed in figure 18, offering a set of building blocks that users could leverage to speed up the development process. This allows online collaboration across users and teams to be supported [66]:

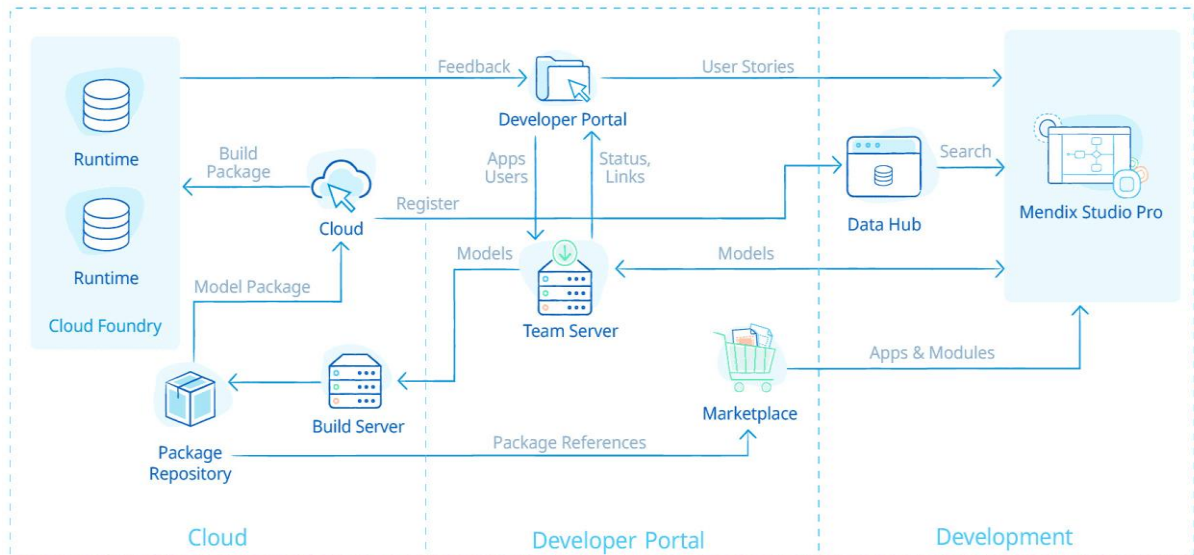


Figure 18 - Key Components of the Mendix Platform Architecture (Source: [66])

By analysing the diagram shown in figure 18, it is possible to split the components into three main groups (Cloud, Developer Portal and Development [66]):

Cloud (Backend infrastructures and services that support app deployments)

- Cloud Foundry
- Package Repository
- Cloud

Developer Portal (Tools that ease team collaboration, app management, and monitoring)

- Developer Portal
- Team Server
- Marketplace

Development (Focuses on tools that developers use to create applications)

- Studio Pro
- Data Hub

The Mendix development cycle starts in **Mendix Studio Pro**, where developers create the application models, including if necessary pre-built Apps & Modules taken from the **Marketplace**.

These modules are then stored in the **Team Server**, enabling collaboration and version control. The **Developer Portal** supports the development process in tracking the progress, managing User Stories, and solicitations for Feedback.

The **Data Hub** is employed for the purpose of connecting the application with external data sources and integrating it for seamless access to data. Once the app is set and ready for deployment, a **Model Package** is created from **Mendix Studio Pro** and then sent to the **Build Server**. Here, the build server constitutes compilation for the app into a **Build Package** and then displays it in the **Package Repository**.

Afterwards, it is deployed on the Cloud using **Cloud Foundry**, where it will run in the Mendix **Runtime** environment.

Finally, after implementation, the app is going to be monitored, and feedback will be gathered through the **Developer Portal** to aid in the informing cycle of development and updates [67].

[This page has been intentionally left blank for formatting purposes]

Technical, Cost, and AI Comparison

Chapter Introduction

OutSystems and Mendix both share a key-similarity: they serve as multi-purpose low-code platforms for enterprise application development. However, they go radically different ways in their approach. While OutSystems is actually built to make developers more productive, Mendix bridges the gap between business and IT. Whereas OutSystems underlines developer independence, Mendix tries to support domain experts and software developers collaborating jointly on the same application. OutSystems focuses on software developers, while Mendix targets its offerings to citizen developers [68].

The companies' main goals, since they were founded, are displayed in figure 19, where it is possible to identify the different values, they stand for:

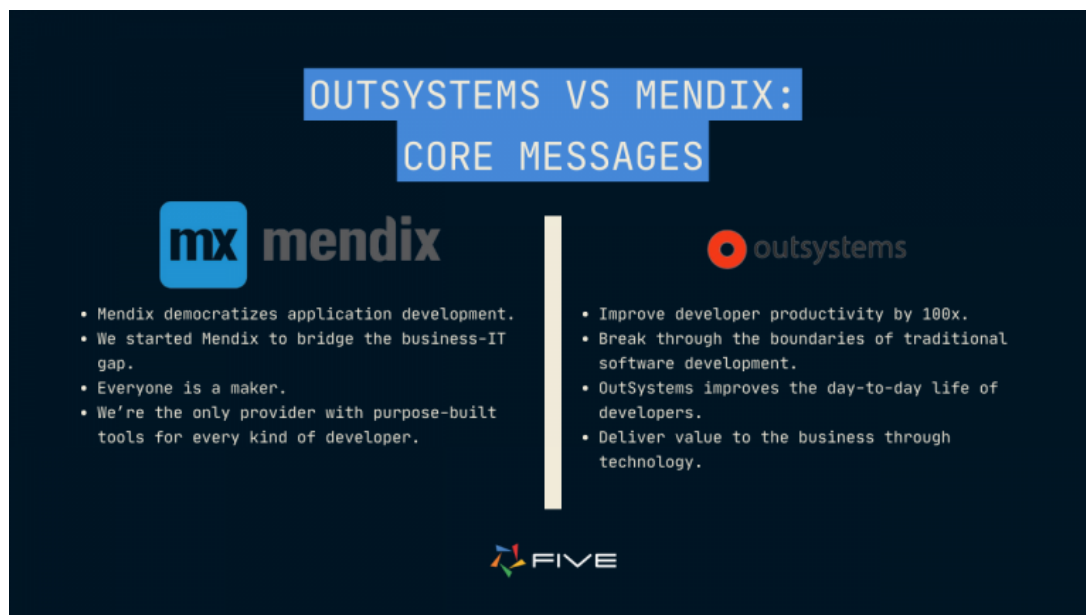


Figure 19 - OutSystems vs Mendix: Core Messages (Source: [68])

As such, the answer to the question of which one is better, can be hard to determine since both strategies have their positive and negative aspects, with OutSystems ahead in market comparison.

This chapter will mostly focus on a technical, pricing and artificial intelligent tools comparison between these two platforms, with an attempt to demonstrate the low-code tools each one offers and how it differs from the other.

4.1. Architecture & Interface

Mendix

Mendix is built in scala/java and runs in the JVM, which makes it ideal for companies with an existing java infrastructure [69]. Mendix focuses on enterprise landscapes, integrating collaboration tools that connect development teams with business users. It streamlines the development lifecycle from design to deployment and provides a tightly integrated cloud-based environment [70]. Overall Architecture:

- **Cloud Services** handle hosting, scaling, and performance across different environments;
- **Developer Portal** does consolidate project management, which supports collaboration and lifecycle tracking of projects.
- **Development Environment:** It wraps visual modelling with the freedom of custom code for building scalable enterprise applications. Combined, they introduce a smooth manner of creating, deploying, and managing applications.

Mendix Studio Pro, depicted in figure 20, is a visual IDE that accelerates application development, automates routine work, and helps the user to handle complex tasks with ease, with customizable themes, drag-and-drop functionality, reusable components, and full-stack capabilities [72].

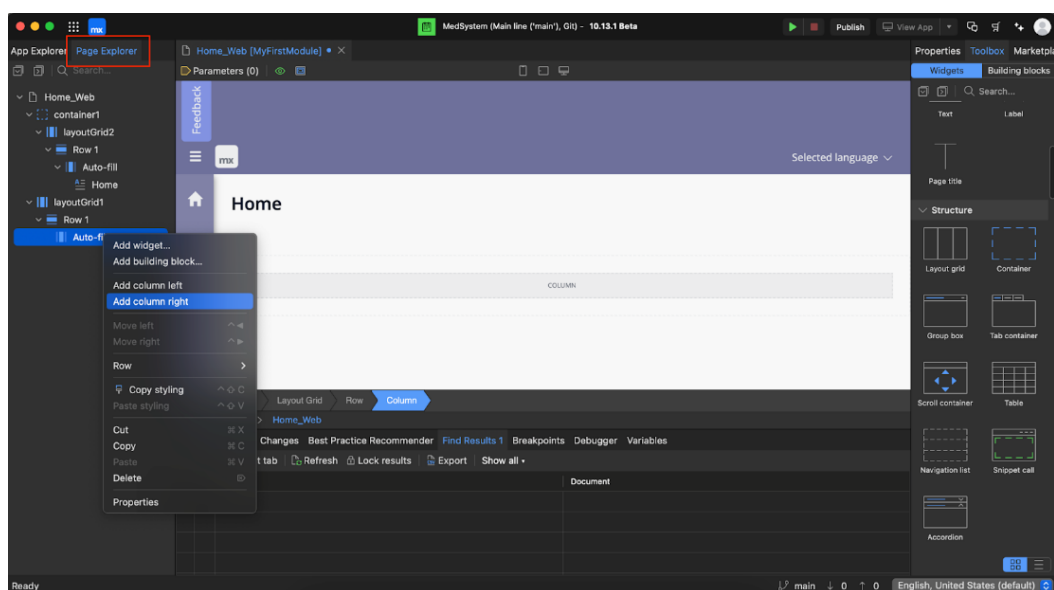


Figure 20 - Mendix Studio Pro IDE

Some important qualities of Mendix Studio Pro are:

- **Intuitive UI;**
- **Smart workflow automation;**
- **Scalable application logic;**
- **AI-assisted development;**
- **Seamless data integration;**
- **Version control and multi-user development;**
- **Built-in Mendix Marketplace;**

OutSystems

On the other hand, OutSystems' extensions to the platform at the server side can be written in **C#** and extensions to the front end can be written in **JavaScript** and the platform is built to provide a highly flexible cloud-native architecture for enterprises seeking both rapid development and robust scalability [71].

The OutSystems platform has a layered ecosystem, from the top comprising all the required tools, components, builders, and repositories for the proper management of SDLC to an AI-powered services layer at the bottom that contains all the tools to eradicate repetitive tasks by automating complex change management, code generation, integrations with third-party enterprise systems, etc.

OutSystems is built on a layered ecosystem in which all the tools and components required to manage seamlessly the Software Development Life Cycle are provided. At the top layer, it offers:

- **Rich set of development tools, repositories, and builders** that support each stage of application development, from designing through deployment [70].

All the way at the bottom, OutSystems has placed:

- **AI-powered services layer** that automates many of the repetitive and complex tasks associated with development. It enables automated change management, code generation, and integrations with third-party enterprise systems [70].

This layered ecosystem makes OutSystems' vision a reinforcement of rapid development and deployment, enhancing efficiency while sustaining scalability and flexibility for enterprise-grade applications.

Service Studio is the OutSystems Low-Code and visual development environment, it is a unified IDE that integrates visual development with the ability to extend using C# and JavaScript. It provides a single tool for both beginner and professional developers (available in Figure 13) , that allows them to [71]:

- **Create applications and Modules on the server;**
- **Create the user interfaces for Traditional Web, Reactive Web, and Mobile Applications;**
- **Define the data model;**
- **Define business processes and timers (batch processes);**
- **Debugs apps;**

4.2. Database & Entity relationship

Mendix

Every Mendix application requires a functioning database to be operational. During the development process, Mendix will automatically establish its Built-In database for the application, which is a flat file database called HSQLDB (HyperSQL DataBase) [73].

According to Mendix, this database is a leading SQL relational database software written in Java. It provides a compact, rapid, concurrent, and transactional database engine that includes both in-memory and disk-based tables and is compatible with both embedded and server modes. This consists of a robust SQL tool for the command line and easy-to-use query tools in a graphical user interface.

Despite having a Built-in database, Mendix allows the dev-user to choose the desired database, so it supports the following options:

- **MYSQL database**
- **Microsoft SQL Server**
- **Oracle database**
- **SAP HANA**

And for applications deployed to Mendix Cloud, Mendix uses a **PostgreSQL database** for storing the data defined in the app domain model (or models). In figure 21 it is possible to identify the different database options available at the application configuration settings:

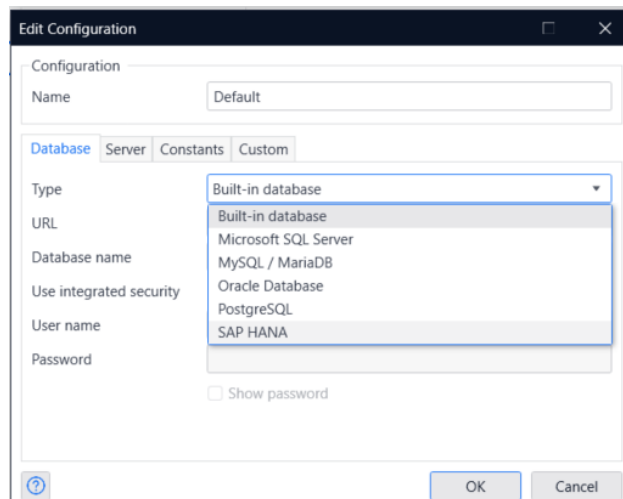


Figure 21 - Different database options available at the application configuration settings

In Mendix the database is modelled using the Domain Model. Similar to an ERD (Entity Relationship Diagram). It not only generates a practical database based on the Diagram during app compilation, but also provides a command line SQL tool and user-friendly GUI query tools.

A domain model is a diagram representing a set of entities and relationships and if we compare it to conventional SQL, a table is similar to an entity. Entities have attributes, which are comparable to the fields, and use fundamental data types, for instance, string, integer, and date-time. An entity may have associations to other entities as one-to-one (1:1), one-to-many (1:*), or many-to-many (*:*) relationships [74].

OutSystems

OutSystems uses an SQL database (Microsoft SQL Server or Oracle) as a native database, but it can also integrate with external databases depending on how the applications are deployed, but it is also useful for businesses since it can integrate with already existing databases. For **on-premises** applications in OutSystems, the supported databases are [75]:

- **Microsoft SQL Server** (Default database);
- **Azure SQL database**;
- **Oracle database**;
- **IBM DB2** (less commonly used);

For **cloud deployments** in OutSystems, the platform supports several cloud-based databases, depending on the external chosen cloud provider. The main databases used in cloud environments are:

- **Microsoft SQL Server:** (Default database) for applications hosted on OutSystems Cloud;
- **Azure SQL Database;**
- **Amazon RDS (Relational Database Service)** that support database engines like (SQL Server, PostgreSQL, MySQL);
- **Oracle Cloud Infrastructure (OCI);**
- **Google Cloud;**

In OutSystems Service Studio an entity represents the logical structure of a business object, including its relationships to other entities and in order for two entities to be related, a reference attribute is created. It should correspond, conceptually, to what would be called a "foreign key" in database terminology. Like in Mendix Domain model entities can be related to each other in one-to-one (1:1), one-to-many (1:*), or many-to-many (*:*) relationships.

In the case of Many-to-Many, the dev-user have to create a third entity that contains at least two reference attributes, for representing the relationship from Entity A to Entity B. Actually, a Many-to-Many relationship is existing between two One-to-Many relationships through that third entity, where both entities can link through it. This way, the user will have an organized way of handling data object relationships in OutSystems [76] (This example will be approached in the following Use-Case).

Data Model Use-Case

In figure 22, it is depicted a **Use-Case**, which was designed to represent a sample design of a scalable data model for E-commerce Orders and Reviews, it will also be used to compare a Mendix built data Model with one built in OutSystems Service Studio.

The example given aims to highlight the two types of Entities used in Mendix which can be:

- **Persistable:** Is an entity whose data is stored permanently in the database. This type of entity is used to model data that needs to be saved and retrieved across sessions, such as user accounts, orders, or product information. In my Use-Case example, the blue entities are the Persistable ones which share different connections among them.

- **Non-Persistable:** Are ideal for storing temporary or transitory data, like user inputs or calculations that don't need to be stored in the database. In my Use-Case example the non-Persistable entity is the **OrderSummary** which is represented in yellow and since it stores temporary data, is not connected to other entities.

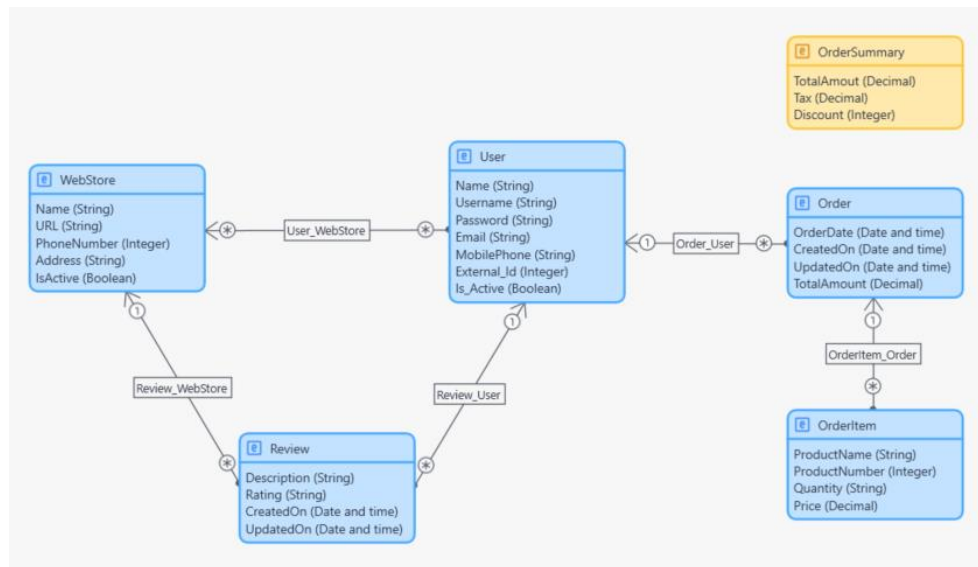


Figure 22 - Design of a scalable data model for E-commerce Orders and Reviews using Mendix

In Mendix, the association modelling is more granular, since every time a connection is established between two entities, the system creates separate associations for each one (e.g: **Order_User**, **Review_User**) enhancing a greater visibility and control over each individual association, allowing for more complex, custom relationship structures if needed.

Figure 23 represents the Entity Diagram created using OutSystems Service Studio, with the same goal as the data model built above.

In contrast with Mendix, in OutSystems associations are relatively straightforward, the system generates an association table behind the scenes, but this is abstracted from the developer, who does not see a separate association per relationship unless it is explicitly created.

Finally, regarding to the handle of a many-to-many relationship between two entities (e.g: **User** and **WebStore**), OutSystems automatically creates a single association entity (like Entity **Review** in this case) to manage this relationship. This approach allows association entities to be more flexible when building the model. If the user knows that the relationship between two entities may evolve, creating the association entity and adding a new field by extending them is pretty simple instead of changing the core of the entities (**User** and **WebStore**) themselves.

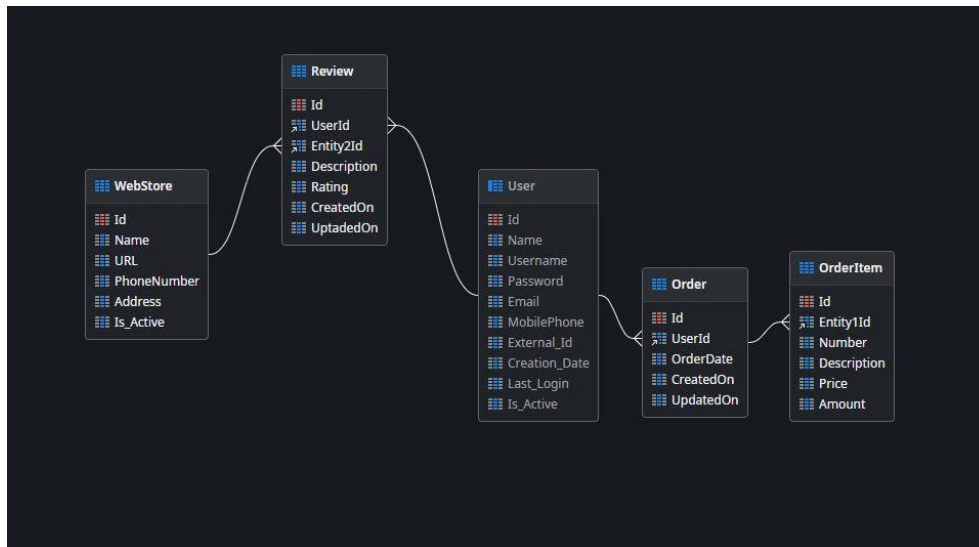


Figure 23 - Design of a scalable data model for E-commerce Orders and Reviews using OutSystems

4.3 Community Support

Mendix and OutSystems are evenly matched in terms of community support, since both have lots of community members who support each other, making it more of a similarity than actually a difference. The deployment of low-code platforms is supported by numerous community members who, constantly, provide solutions for various issues [78].

- **OutSystems** provides a forge repository (**Appendix D**) containing reusable components, UI elements, open code modules, connectors, frameworks, and other resources for developers. The user interface, integrations, device capabilities, functionalities, and utilities classify these assets and currently there are 7138 available assets [77] .

- . **Mendix** provides a Marketplace with templates, modules, connectors, pre-built components, and widgets (**Appendix D**) to assist in expanding application capabilities and speeding up the development process. Developers have then the option to browse the marketplace based on industries, content type, compatibility, tags, and rating [78].

4.4. Error Handling

All applications will face errors eventually, which can greatly affect operations and customer satisfaction and so by incorporating error prevention and proactive monitoring, businesses can guarantee smoother operations and improved customer experience.

Both Mendix and OutSystems offer strong error handling capabilities, but they vary in their approach to handling errors during development and runtime, as well as in how they support developers in troubleshooting and fixing problems.

Mendix – (Error Handling during development and runtime)

Mendix offers compile-time error detection during development, pointing to problems like unresolved attributes or broken microflows, offering a built-in debugger for setting breakpoints and stepping through microflows or nanoflows, from here the user is able to query variable values to track down where any issues originate from [79].

In the case of errors at runtime, Mendix automatically executes rollbacks for microflows and nanoflows. If an error happens during a microflow or a nanoflow—some operation on a database, for instance—Mendix will roll back changes done within that transaction, so no partial data or inconsistent data state is committed. For the rollback, there are 4 error handling options that are displayed in figure 24 [79]:

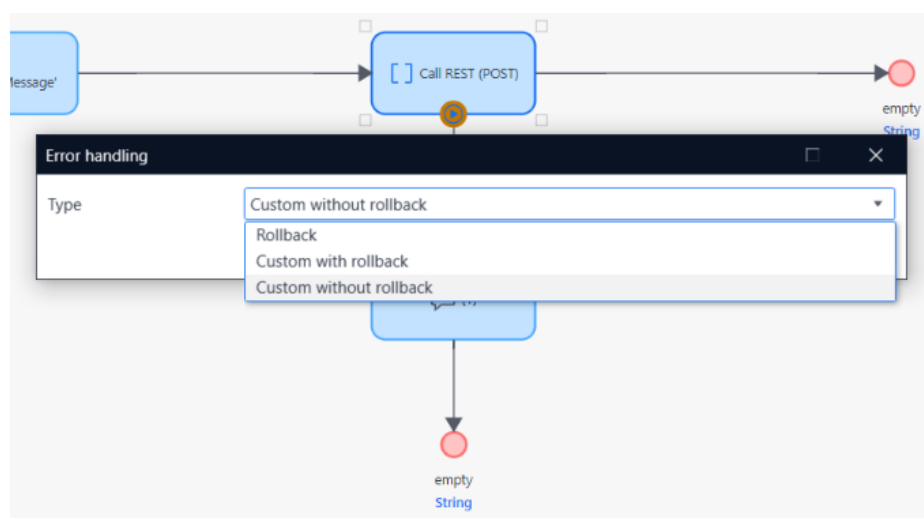


Figure 24 - Error handling options for Microflows and Nanoflows in Studio Pro

- **Rollback (default)** – If the microflow is not executed successfully, it will automatically rollback the transaction. It commits no changes when an error occurs and keeps data consistent.
- **Custom with rollback** – The user can deal with an error manually, performing custom logic, the framework still manages the rollback of the transaction. Useful when some custom error handling is needed without losing the rollback safety.

- **Custom without rollback** - When an error occurs, it doesn't roll back automatically. This property comes in handy when a developer needs to decide to commit or rollback a transaction based on particular conditions.
- **Continue** - This option continues the execution of the microflow even if an error occurs, without rolling back the transaction. It's typically used when errors are not critical and can be ignored, or when compensating actions are defined later in the microflow.

OutSystems – (Error Handling during development and runtime)

During development in OutSystems, several mechanisms proactively prevent errors. Indeed, the most prominent one should be the **TrueChange Technology** exemplified in figure 25, since it gives feedback on the changes of apps in real time. TrueChange includes AIFusion, an OutSystems AI service that automates complex manual tasks, validates architectural issues and dependencies, offers architecture management, and oversees real-time performance [80].

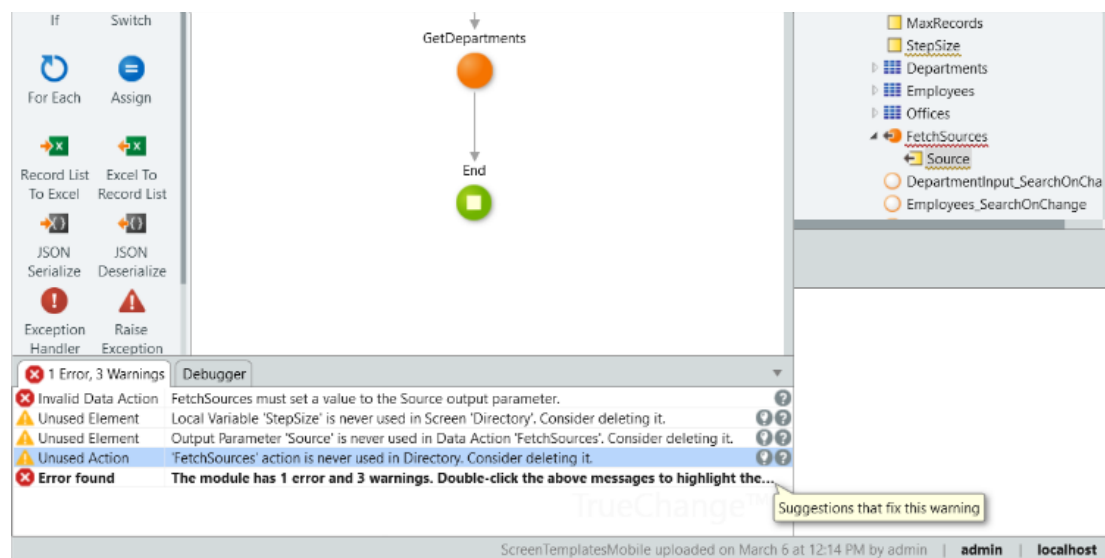


Figure 25 - TrueChange technology live working in OutSystems IDE (Source: [80])

In figure 25 it is depicted a demo of how TrueChange works, and as it is possible to visualize, it aims to minimize technical debt by simplifying and streamlining app modifications, which enables the team to release new versions or updates more quickly. And since it is integrated into the OutSystems development platform, there is no requirement to set up complicated tools for configuration.

When creating an application, the users strive to follow business logic, requirements, and best practices for performance. However, unforeseen interactions by users, browsers, networks, or servers can cause exceptions but in OutSystems runtime error handling is robust, with a strong focus on exception management [81]:

- **Built-in Exceptions:** OutSystems comes with a broad set of built-in exceptions to allow handling common error situations, whether at the database or communication level. These exceptions enable the developer to create specific error handling for situations that require special handling.
- **Custom Exceptions:** Developers define and use custom exceptions depending on the more specific business logic. These can be thrown and managed at key places in the application flow to further tune control over error responses. (e.g: Figure 26 illustrates a User-Exception use-case where, instead of halting user interaction, it offers an alternative route by showing messages on the screen to provide feedback to the user [81]).

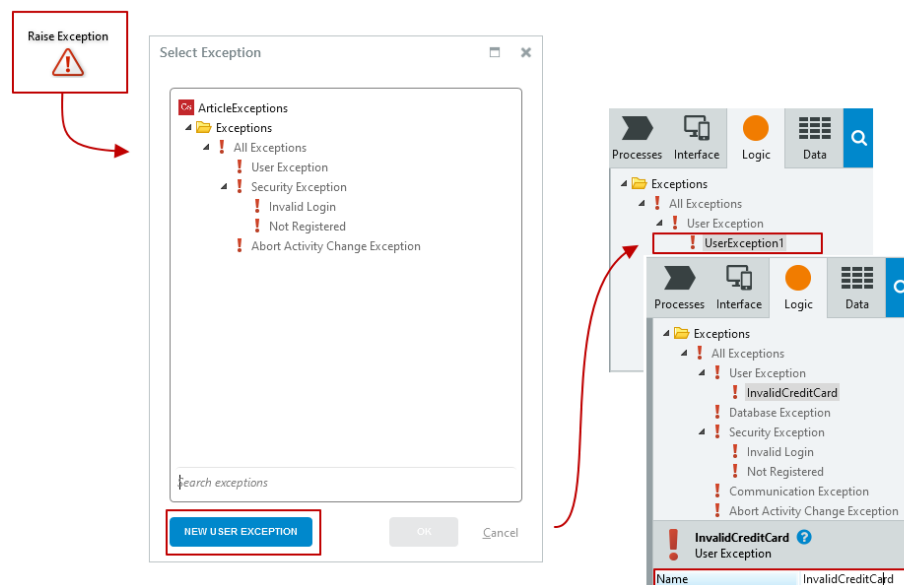


Figure 26 - User-Exception use-case scenario (Source: [81])

Additionally, OutSystems enhances error management also with other various tools and features:

- **Error Logs;**
- **Retry Mechanisms;**
- **Error Screens;**
- **Retry Mechanisms;**

These features guarantee that OutSystems applications are both strong and dependable while also upholding a high level of user happiness by implementing efficient error prevention methods while running.

4.5. Integrations

One of the critical differentiators for UI-centric low-code platforms is integration. This is the case because, for many products, there are no tools to link up a new web front-end or new mobile with core IT applications and databases. Therefore, choices differ among enterprises, developers with non-technical backgrounds, and so on.

Where in Mendix, citizen developers prefer a comfortable visual experience, on the other hand, OutSystems offers enterprises an advisable way of controlling IT application integration and a proper way of collaboration with citizens in the case when the two development groups collaborate.

Both Mendix and OutSystems can exult an unlimited number of integrations besides robust community-built components and apps [82] and while Mendix provides a wide variety of third-party integration options, OutSystems excels in this aspect since OutSystems provides integration with over 400 out-of-the-box systems as a built-in feature [83].

Examples of platforms that OutSystems can integrate with:

- **Salesforce**
- **QuickBooks**
- **UPS**
- **PayPal**
- **Okta**
- **Facebook**
- **Twitter**
- **LinkedIn**
-

Examples of platforms that Mendix can integrate with:

- **Salesforce App Cloud**
- **SAP CRM On-demand**
- **Tableau Software**
- **GitHub**
- **Skype**
- **Microsoft Excel**

Integration-Oriented application Use-Case (AI Gateway)

The following use case **scenario** was designed to demonstrate how integrations work both in OutSystems and Mendix. In this scenario, the focus was on integrating two applications with a Representational State Transfer (REST) service provided by OpenAI, with the main goal of implementing an AI chatbot similar to ChatGPT. This integration highlights each platform's capabilities in connecting with external services and delivering AI-driven solutions within enterprise applications.

REST Service

REST is a software design that sets rules for how an API operates. REST was first developed as a set of rules for handling communication on a complicated network such as the internet, its architecture can be used to improve efficient and dependable communication on a large scale. The user can effortlessly put into action and make changes to it, enhancing visibility and ensuring compatibility across different platforms for any API system [84].

A REST service enables applications to interact with each other via the HTTP protocol on the internet, and strengths like being straightforward and expandable is what makes them commonly used for merging various systems due to their adherence to a specific set of principles.

Resources in a REST service are accessed through Uniform Resource Locators (URLs), and their interaction is carried out using HTTP methods like [84]:

- **GET:** Used to retrieve data from the server;
- **POST:** Used to send data to the server;
- **PUT:** Used to update existing resources on the server.;
- **DELETE:** Used to request to remove resources from the server;

The other key elements used in RESTful APIs and HTTP communication in general as they allow clients and servers to exchange additional information during a request and response are [84]:

- **HTTP Headers:** Headers in a request represent the data shared between the client and server during communication. As an example, the request header specifies the format of the request and response, including details about the request status.
- **HTTP Parameters:** HTTP parameters offer information that the server can utilize for handling requests. These are frequently used in API requests and can be transmitted through different ways like (Query-Parameters, Path-Parameters, Body-Parameters).

Finally, regarding responses, in a REST service they are commonly given in formats such as **JSON** or **XML**, simplifying integration throughout different platforms and coding languages. These services are frequently used to link front-end applications (such as websites or mobile apps) with back-end systems or to incorporate third-party services, (e.g.: AI tools) just like in this Use-Case.

OpenAI developer platform – Chat Completions

The two applications named “**AI Gateway**” developed using Mendix Studio Pro and OutSystems Service Studio focus on an integration with the OpenAI REST API, more specifically the Chat Completions endpoint.

The Chat Completions endpoint is designed to handle conversation-like interactions, where it generates contextually appropriate responses based on the user’s input as it processes the incoming text (Request) and delivers a relevant, coherent reply (Response), leveraging AI models like GPT-4, GPT-4-Mini and GPT-3.5 Turbo [85].

When making a request to the Chat Completions endpoint of the OpenAI API, there are 2 important HTTP headers that need to be used to ensure the request is properly formatted and authenticated, and should follow a specific format :

- **Authentication Header:** [Authorization: Bearer <API-KEY>](#)

(The Bearer token is used as a security measure that ensures only authorized users with a valid API key can access the API, and the API-KEY is a confidential key, used either at a user's or service account's level, which grants entry to either one project or all organizations and projects the user is part of)

- **Content Type Header:** [application/json](#)

(This specific header is required for the server to correctly understand the incoming request, as the request body must be in a JSON format.)

In the following image 27 and image 28 it is represented the Post method URL used 'https://api.openai.com/v1/chat/completions' as well as the Request structure used for this use-case both in OutSystems and Mendix, whose parameters will be precisely explained below:

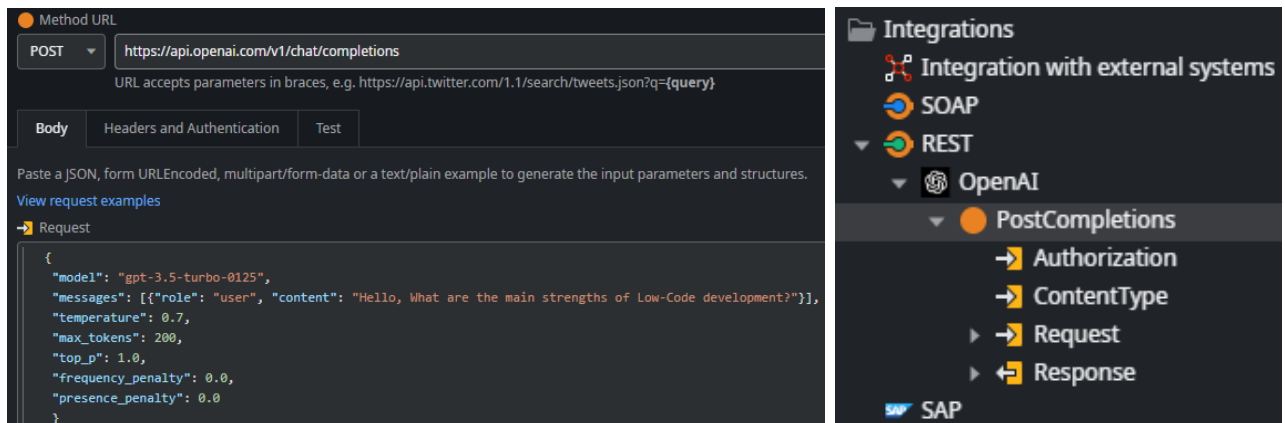


Figure 27-Request body of the Post Completion method defined in the exposed REST API using OutSystems



Figure 28 - Request body of the Post Completion method in the exposed REST API using Mendix

Request body [85]:

- **Model:** `gpt-3.5-turbo-0125` (Specifies the AI model to be generate the response);
- **Messages:**
 - Role:** `user` (Defines the speaker in the conversation, can be a user or an assistant);
 - Content:** `"message"` (The message or prompt sent by the user);
- **Temperature:** `0.7` (Sampling temperature range, from 0 to 2, values closer to 0.8 will result in a more random output, whereas values closer to 0.2 will create a more deterministic outcome);

- **Max-tokens:** 200 (Sets the maximum length of the AI's response in tokens which can be as short as one character or as long as one word);
- **Top-p:** 1.0 (Ranging from 0 to 1, a value of 1 turns off nucleus sampling, enabling the model to consider every potential token. This implies that there are no limitations on the distribution of token probabilities, which could result in a wider range of responses);
- **Frequency-penalty:** 0.0 (Ranging from -2.0 to 2.0, lowers the chance of the AI repeating the exact same lines. A 0.0 value indicates that no penalty is enforced, allowing the model to repeat phrases if necessary);
- **Presence-penalty:** 0.0 ((Ranging from -2.0 to 2.0, lowers the chance of the AI repeating the exact same lines. A 0.0 value indicates that no penalty is enforced, allowing the model to introduce new topics if necessary);

Roles

In **OutSystems** besides the default Anonymous and Registered role, it was also created a new role called Administrator, with full access to all the app functionalities. In **Mendix**, the Administrator role was also created as an app user role in the application security menu, granting this user access to modules as a whole, as it is possible to verify in figure 29:

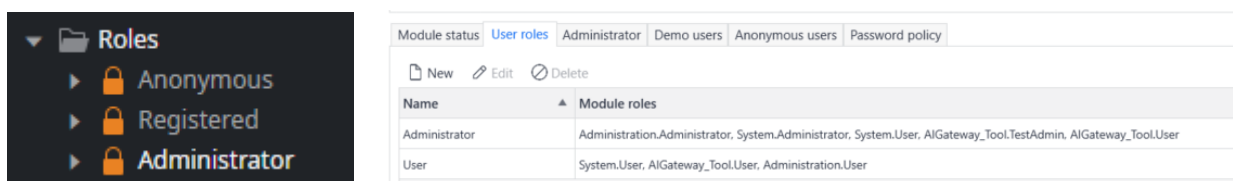


Figure 29 - User roles defined in OutSystems and Mendix respectively

Development using OutSystems:

For the AI Gateway application in OutSystems, it was necessary to create 2 different modules, since one responsible for managing the integration with external systems (**GPT_Connect_IS**) and other module (**AI_Gateway**) responsible for the user interface (UI) and user interaction with the chatbot functionality containing the screens, forms, and user experience logic to interact with the OpenAI-powered chatbot.

In the images below there are two important actions depicted:

- **CommunicateOpenAI:** Represented in Appendix E (3rd) as a **Server Action**, it runs on the server and typically handles more complex tasks like database access, heavy processing, or like in this scenario, external integrations. It sets variables, makes a POST request to the REST API ("PostCompletions") and then stores and processes the output before sending it back to the client.
- **ButtonOnClick:** Displayed in Appendix E (2nd), this action is a **Client Action** meaning that it runs on the user's device (in the browser or on a mobile app) and interacts with the UI in real-time. It runs logic such as disabling local UI elements, sending a request (calling the server action), and then re-enables UI elements after receiving a response from OpenAI.

Finally, in the appendix E, it is also possible to verify how the application behaves live, with the admin (OSAdmin), the one logged trying to communicate with the ChatGPT, and in the last image it is possible to verify that the integration ran well.

Development using Mendix

The AI Gateway application developed using Mendix, it was necessary to create a JSON structure file to save the request body and another to save the response body.

Along with these 2 JSON files, to accomplish a correct integration as shown in Appendix F, it was necessary to create one **Import Mapping** (available in Appendix F), which is used to specify how incoming data from external systems or APIs, (in this case used to process the response body structure) is transformed into my Mendix app's domain model or entities.

And one **Export Mapping** (available in Appendix F), which involves defining the structure and format in which data from my Mendix app's domain model is sent to external systems (in this case used to process the request body structure).

In the images below, are represented the two most important microflows used to implement the application's logic:

- **ACT_PostCompletions:** The ACT_PostCompletions microflow, available in figure 31 prepares the data of the API call. It takes as input the request and response parameters, builds the root object, and then calls the DS_CallRest_Post microflow to perform the API call. Finally, it handles the server response.

- **DS_CallRest_Post:** This microflow sends the API request to OpenAI. It uses a pre-prepared root object, sends a POST request, and subsequently handles the response. It retrieves the 'Message' from the API's response and shows the result to the user, as it is possible to see in figure 30.

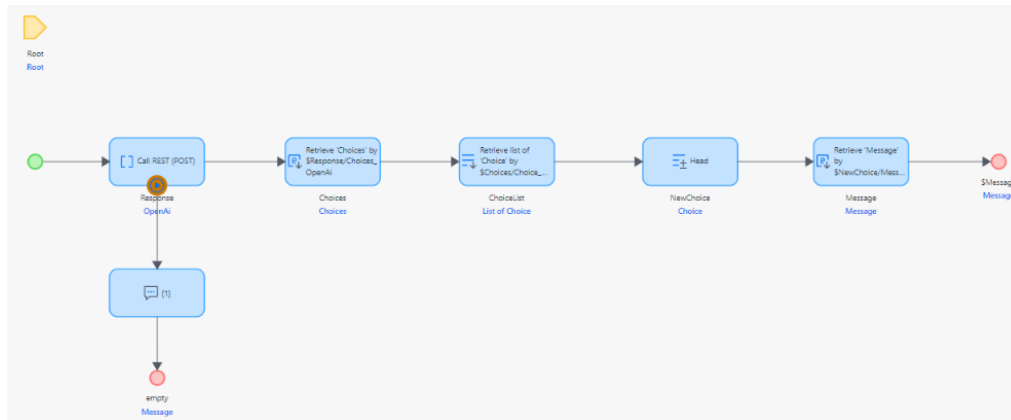


Figure 30 - DS_CallRest_Post microflow logic

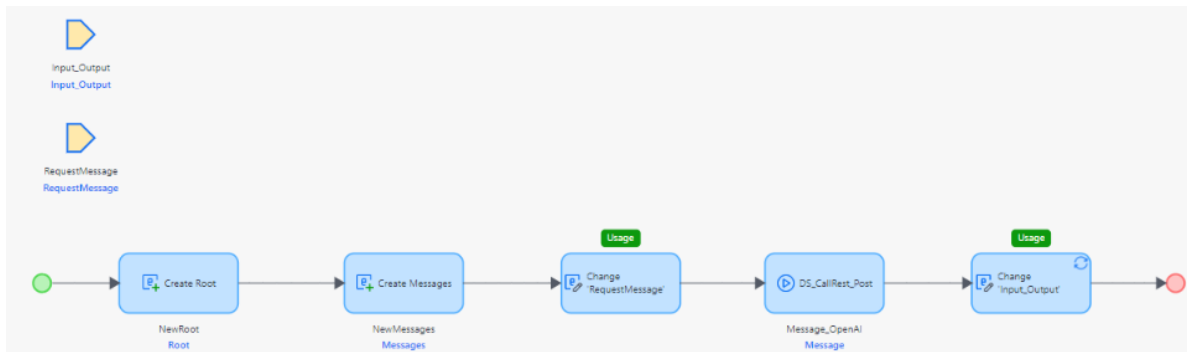


Figure 31 - ACT_PostCompletions microflow logic

Ultimately, regarding the integration process with an external Open AI API it is possible to get some conclusions approached in table 6.

Table 6 - Integration Use-Case findings comparison

Mendix	OutSystems
- Mendix offers a highly visual and guided integration approach with the use of import and export mappings. - In the platform, the user can map external services and data structures directly into its domain model, allowing the integration to be more intuitive for complex structures such as JSON or XML. - The data transformation and validation capabilities of Mendix ensure that incoming and outgoing data is also perfectly aligned with the application's schema.	- With OutSystems, any API is easily integrated, basically, any external web service can easily be consumed with the built-in REST and SOAP connectors. - The focus of the tool is generally more on rapid integration with minimal configuration, especially when consuming standard web services. - While OutSystems can be fast integrating APIs, it also requires more manual work in cases of complex data transformations, as it has its focus set on speed rather than exhaustive customization.

4.6. Pricing plans

4.6.1. OutSystems pricing model

The wider the development is with OutSystems, the more reusable capabilities users will create, leading to quicker development of future projects. Whether expanding upon a first resolution or building something entirely independent.

To achieve this, OutSystems adjusts the license based on three different key metrics [86]:

- **Number of Internal Users:** Employees are licensed in batches of 100, with the initial license covering the first 100 users. They are named by the email domain of the organization and as the numbers grow, additional packs of 100 internal users can be purchased. User management is flexible and can be integrated with domain servers, or manually managed through the tools provided by OutSystems, so in this way, internal users within the system are easily controlled.

- **Number of External Users:** External users, such as customers or partners, are sold in packs of 1,000 and are differentiated by email domains different from the internal users. Only those logging in are counted toward the license, not anonymous users who access public-facing pages without logging in. To save money, companies usually program the deactivation of accounts that remain idle beyond a certain period to keep the number of licensed users approximately correct.
- **Number of Application Objects (AOs):** Application objects represent the technical complexity of an application, including screens, database tables, APIs and other resources. A small application uses around 50-100 AOs, while larger, more complex apps fall in a range from 500 to 700 AOs. OutSystems supports maximum reusability of components across multiple applications, reducing the AO count and allowing businesses to build scalable solutions without additional licensing costs.

Licensing Plans

OutSystems offers three main pricing models, that may vary in pricing according to the metrics that were specified before and also according to the infrastructure used (cloud and on-premises):

Free Personal Environment (OutSystems 11): This option is an ideal opportunity for students and developers who want to gain hands-on experience with low-code development and also learn and experiment the platform without any cost [87]:

Key-Features:

- The OutSystems Cloud hosts the free personal environment, so students do not have to be concerned with setting up or managing their own infrastructure.
- This environment is specifically intended for development and testing, meaning that apps cannot be used for production purposes.
- Developers will have access to the OutSystems IDE (Integrated Development Environment), including Service Studio for building apps as well as Pre-built Components and Templates [87].

Limitations:

- Includes a restricted amount of internal users (max 100 internal users) and does not offer assistance for external users, making it appropriate for personal use or very small team collaborations.
- An app developed in the free environment is restricted in its complexity because of the existing Application Objects (AOs). A small application usually requires 50-100 Application Objects (AOs), but larger projects will require an upgrade to a paid plan.
- The community support is the only feature available in the free plan, so users must depend on forums and documentation for assistance, without any access to direct professional support from OutSystems [87].

Paid Enterprise Plans (OutSystems 11): The enterprise plans for OutSystems 11 are either on-premises or self-hosted, and pricing starts at about **\$1,513 monthly**, this pricing scales with internal users, external users, and Application Objects (AOs) that measure the complexity of the app. It is ideal for businesses that require on-premise deployment but need more manual work to manage their infrastructure [88].

Key-Features:

- Enterprise plans, unlike the free plan, support both internal and external users, which are available for purchase in Packages (e.g., 100 internal users and 1000 external users) to accommodate varying levels of user access.
- Enterprise applications may be significantly bigger and complex, accommodating hundreds or even thousands of AOs.
- Enterprise plans offer assistance for standard compliance such as ISO 27001 and other standards, ensuring that applications meet the required security and regulatory obligations [87].

Limitations:

- As you increase the number of users or application objects (AOs), the expense of the business plans can rise dramatically.
- For On-premises deployments customers must handle their own infrastructure, such as servers, storage, backups, and updates, for on-premises deployments, which can lead to increased complexity and expenses [88] .

OutSystems Developer Cloud (ODC): The cloud-native offering starting at \$36,300 per year, offers advanced scalability, automated updates, and comprehensive cloud infrastructure management. It is targeted at enterprises that do not want to manage an infrastructure of their own and need features such as microservices, auto-scaling and better performance for large-scale applications [88].

Key-Features:

- Designed specifically for cloud environments, utilizing a modern, container-based architecture that enhances scalability and availability. Managed by OutSystems, it operates over several availability zones, which grants resilience and performance.
- Includes integrated CI/CD pipelines to simplify the application development process from building to deploying. Automating testing, building, and deployment across environments which accelerates development.
- Unlike the on-premises version, ODC takes care of automatic updates and maintenance. This allows developers to focus just on app development without worrying about platform maintenance [89].

Limitations:

- ODC starts at \$36,300 annually, which is more expensive than on-premises options, especially for small businesses or those with limited budgets.
- ODC is only available on the cloud, making it unsuitable for companies needing an on-premises solution due to security or compliance concerns.

- Roles in ODC are application-specific, so they cannot be used for more than one application. Deploying multiple interconnected applications requires more meticulous role management planning [89].

4.6.2. Mendix pricing model

Mendix provides multiple pricing options, customized to various users and organizations, including small teams and large enterprises, these prices vary according to the plan chosen and also according to the number of applications aimed to be deployed. Figure 32 and figure 33 portray a summary of Mendix main pricing structures:

Show pricing for: One App Unlimited Apps	Free Jump in and give the platform a try Starting at €0/month Get Started	Basic Create simple workgroup applications Starting at €52,50/month Purchase now	Standard Build department-wide business applications Starting at €900/month Contact Us	Premium Build mission-critical core systems Get it all Get a Quote Contact Us
---	---	--	--	---

Figure 32 - Different pricing structure for a "one app" license (Source: [91])

Show pricing for: One App Unlimited Apps	Free Jump in and give the platform a try Starting at €0/month Get Started	Standard Build department-wide business applications Starting at €2,100/month Contact Us	Premium Build mission-critical core systems Get it all Get a Quote Contact Us
---	---	--	---

Figure 33 - Different pricing structure for multiple apps license (Source: [91])

Free Plan: Mendix's Free Plan is an entry-level offering designed for hands-on experimentation by a 1 individual or small team without incurring any cost. It's ideal for prototyping purposes, creating demos, or small apps with limited functionality [91].

Key-Features:

- The Free Plan allows users to utilize no-code and low-code development tools to swiftly build web or mobile apps with visual interfaces.
- The plan consists of two settings for each app: one for development and one for production [91].

Limitations:

- Although users have the option to use apps, the free plan is not appropriate for large-scale production settings.
- The free plan only offers community-based support, meaning users rely on forums and self-help resources. There's no access to professional or 24/7 customer support [91].

Basic Plan: The Mendix Basic Plan is ideal for small teams or organizations that want to build simple applications, which don't bear too much complexity. The pricing for the basic plan starts from €52.50/month for up to 5 users, adding €10 a month for every user added beyond that [92].

Key-Features:

- The Basic Plan provides collaboration tools that allow teams to work together on application development using a visual IDE and it also supports agile project management.
- This plan comes with a 99.5% uptime guarantee, ensuring your applications remain available and reliable for day-to-day operations [91].

Limitations:

- The Basic Plan only offers **2** environments per app, one for development and one for production, which restricts the flexibility for testing and staging environments.
- Assistance is only provided from 9 AM to 5 PM during working hours, which may not be enough for teams that require 24/7 support [92].

Standard Plan: Targeting medium to large organizations that have a need to build and scale via department-wide or business-critical applications, the standard plan offers more features than the Basic but still has limitations. The pricing of a Standard Plan starts with €900/month in case of one app, and €2,100/month for unlimited apps, but it increases greatly once more users or apps are added: for every additional user, it costs an additional €25 per user, per month.

Key-Features:

- The Standard Plan includes up to four environments per app, allowing for separate development, testing, staging, and production environments.
- Organizations benefit from dedicated database tenancy, which provides better performance, security, and data isolation compared to shared databases [92].

Limitations:

- Limited vertical scalability (adding resources to a single server) is more limited compared to the Premium Plan.
- Assistance is only provided from 9 AM to 5 PM during working hours, which may not be enough for teams that require 24/7 support [92].

Premium Plan: the Mendix Premium Plan is targeted for very large-scale enterprise needs, which needs more advanced build and runtime capabilities in order to drive mission-critical applications. The plan grants the uppermost level of support, scalability, and performance, hence, highly suitable for organizations with complex needs [91].

Key-Features:

- The Premium Plan allows horizontal scalability, enabling the addition of more resources to manage higher traffic or workloads. It also has fallback options, guaranteeing that applications stay functional during server outages or peak traffic times.
- Enterprises get access to 24/7 professional support to handle any critical issues at any time. And additionally have a 99.95% uptime guarantee provided [93].

Limitations:

- The Premium Plan comes with **custom pricing**, which can be significantly higher than the Standard Plan [93].
- While the Premium Plan offers a wide range of customization choices and scalability, it also results in increased difficulty in handling the infrastructure [93].

4.7. AI-Assisted Development (AIAD)

The integration of AI with low-code development is one of the most thrilling and promising features of software development.

Automation of tasks, along with AI, amplifies efficiency and boosts innovation as the most important processes in workflow optimization are done automatically by AI, thanks to intelligent features that empower developers' productivity and applications in general [94].

AI-assisted development involves using artificial intelligence and machine learning algorithms to change the way software is developed, so the low-code development platforms using AI tools assist the developers in code generation, error detection and feature recommendations, as a result, these tools accelerated development for innovating faster and building applications that are smarter with less effort [95].

4.7.1. OutSystems AI Tools

OutSystems is dedicated to assisting users in utilizing the latest technology advancements in the industry. Not only through components in the OutSystems Forge that provide the ability to integrate advanced AI and machine learning features into low-code apps, but also through AI-powered tools designed to assist developers throughout the development process.

AI Mentor System:

The OutSystems AI Mentor System is an improvement of the former Architecture Dashboard of the OutSystems Platform. It provides enhanced capabilities and incorporates innovative AI technology which helps IT teams improve architecture performance, detect and resolve security coding issues, reduce technical debt and wasted development time. This solution consists of 5 Mentors depicted in figure 34, providing enhanced capabilities for incorporating innovative AI technology [96].



Figure 34 - AI Mentor system engine (Source: [97])

- **Code Mentor:** AI-driven tool which can predict next steps, propose the optimal code, and generate SQL queries, saving developers a lot of time.
- **Architecture Mentor:** AI ensures application code meets critical architectural standards for scalability and maintainability.
- **Security Mentor:** An AI-based tool that scans code for vulnerabilities, thus helping developers find and fix security risks early.
- **Performance Mentor:** AI does code analytics for performance bottlenecks to make sure applications are always smooth-running;
- **Maintainability Mentor:** AI browses through the entire portfolio for technical debt reduction to identify redundant code and refactoring opportunities.

As an architect, a team-leader or even as a developer, AI-Mentor studio provides a detailed overview of the team's applications' technical debt and thorough examination of the organization's technical debt, allowing users to take the necessary measures to reduce (security, maintainability, performance, architecture and code-quality) risks.

In **Appendix G** is displayed how AI Mentor Studio works, where as a team leader using the tool I can visualize each square representing an app. It shows the app name and the app type. The color of each app shows the user how high or low the technical debt is in that app and technical debt also tells how difficult or easy it is to change and maintain an app or module.

The systems generates a report that can be filtered based on the team, a specific app, module or category, when it was analyzed, the findings status, as well as specific users. To check how to fix the error, the user must click “How to fix” and can also use this information to understand the effort involved in fixing the code pattern error [98].

4.7.2 Mendix AI Tools

Parallel to OutSystems, Mendix also uses AI and machine learning tools to help development teams build applications faster, more consistently, and with better quality. This Mendix's AI-Assisted Development approach is called Mendix AI Assistance (Maia). Maia acts like a virtual co-developer, offering guidance, assistance, and automation throughout different stages of the application lifecycle. Its key features include Maia Chat for real-time guidance in Studio Pro, Logic and Workflow Recommenders, a best practice recommender for in-editor help, and Generative AI tools like the Translation Generator [99].

In Mendix Studio Pro, Maia includes the following capabilities that assist developers throughout different stages of the development process [100]:

- **Maia chat:** The **Maia Chat** tool is a chat interface integrated into Studio Pro that operates on Generative AI. Developers can inquire about app development in Mendix, including the process of implementing concepts, best practices, and development patterns. Constructed using extensive language models (LLMs), the platform draws information from a variety of sources such as Mendix Documentation, Community posts, and Academy learning paths. An example of the application of Maia chat is described in figure 35:

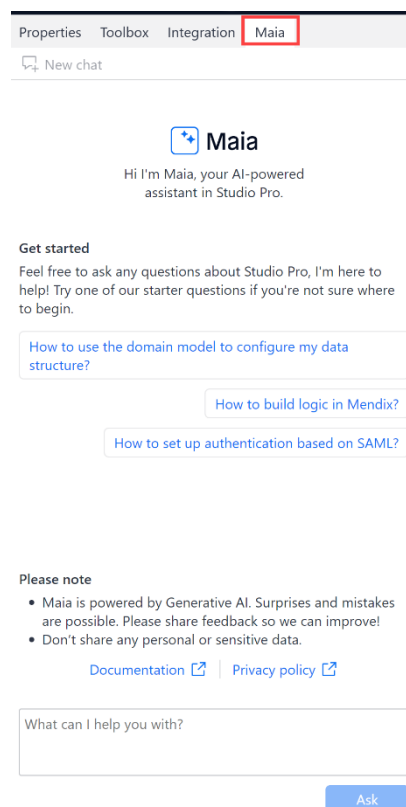


Figure 35 - Maia chat interface used in Studio Pro (Source: [100])

▪ **Best Practice Recommender:** The **Best Practice Recommender** is a smart virtual assistant that assists applications by analysing the app model according to Mendix development best practices in Mendix Studio Pro. It identifies anti-patterns throughout the design and development process, directs the user to these anti-patterns, provides suggestions on how to address them and may in certain instances automatically correct these problems [100]. Image 36 is a display of how this recommender’s pane presents the anti-pattern warnings in Studio Pro.

Inspect now		Limit to current tab	Limit to uncommitted	Export Configuration		
	Category	Code	Message	Element	Document	Module
▼	Performance	MXP002 (1)	Unused Calculated Attribute			
🔍	Performance	MXP002	Unused calculated attribute 'MyFirstModule.Entity.Name'	Attribute 'MyFirstModule.Entity.Name'	Domain model	MyFirstModule
▼	Performance	MXP012 (1)	Microflow with a repeatedly committed variable			
🔍	Performance	MXP012	The microflow 'MyFirstModule.Microflow' has a repeatedly committed variable 'Entity'	Commit object(s) activity 'Commit 'Entity''	Microflow 'Microflow'	MyFirstModule

Figure 36 - Best Practices Recommender Pane (Source: [100])

▪ **Logic Recommender:** The **Maia Logic Recommender** uses machine learning from over 12 million anonymized microflows to identify best practice patterns in application logic. It suggests to the user the top 10 next best actions within the editor by providing contextual recommendations based on the logic’s position, offering search-based suggestions for quick access to actions and auto-configuring parameters for quicker development. These features help both new and advanced developers to improve their speed and accuracy during app development [99]. In image 37 it is possible to identify the logic recommender dialog box presenting the user with 10 contextual suggestions:

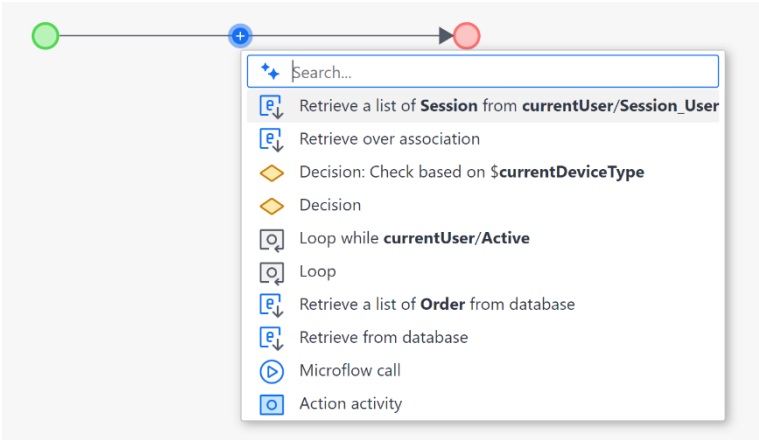


Figure 37 - Logic Recommender dialog box (Source: [100])

▪ **Workflow Recommender:** The **Workflow Recommender** in Studio Pro acts as a virtual co-developer that offers recommendations for the next best activity in workflows. It features smart search to quickly find elements, auto-configuration to pre-populate parameters, and contextual suggestions by analysing the workflow's structure and the page from which it is called [100]. In figure 38 is depicted the workflow recommender dialog box with recommendations of workflow activities to insert.

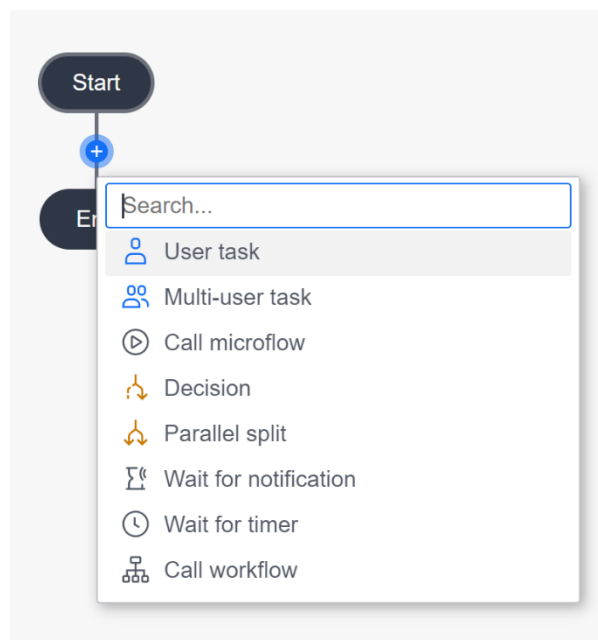


Figure 38 - Workflow Recommender dialog box (Source: [100])

Conclusions and Future Research

5.1 Conclusions

The main goal of this dissertation was to explore the disruptive role of Low-Code development platforms technologies, focusing on how they are reshaping software development.

Therefore, this dissertation examines the importance of Low-Code market leaders (OutSystems and Mendix) in software development, digital transformation, and business innovations. It provides information on how these platforms help organizations create scalable, secure, and strong applications faster and with fewer resources by analysing architecture, development features, integration capabilities, community support, pricing models in detail and the most recent embracing of new AI tools.

Some of the key conclusions drawn from this study:

- **Reduced Time & Cost:** Based on industry reports and real-life examples, Low-Code platforms have actually decreased the time and money required for application development.
- **Enhanced Collaboration:** OutSystems and Mendix enable improved teamwork between technical and non-technical users through their user-friendly visual interfaces.
- **Skill Gap Reduction:** By reducing the need for manual coding and offering pre-built modules, these platforms close the skill gap and accelerate time-to-market.
- **AI-Assisted Development:** Both AI-Assisted Development Tools OutSystems AI Mentor System and Mendix's Maia offer enhancements to the feature development process by providing automated suggestions.

Additionally, despite similar in their goals they too slightly differ in their approaches, as stated in the thesis, there were also differences in the pricing structures: OutSystems is based on AOs and Users, whereas Mendix provides more flexible plans, so it is important that before choosing their path, businesses precisely consider their requirements, customer base, and deployment structure.

In conclusion Low-Code will continue to revolutionize software development with agility and quicker delivery, and as AI keeps improving, it is expected that these platforms will keep getting stronger and provide automated insight, widen the applications in various industries, hence consolidating even more their importance in the evolution of digital technologies.

5.2. Future Research

Moreover, a future research would definitely encompass new Artificial Intelligence tools and algorithms that can be integrated into Low-Code platforms, since nowadays new AI tools are delivered in a much faster pace and it would also be interesting to study in more depth the way sophisticated AI algorithms in the future will be able to automate even greater aspects of software development.

This dissertation also creates a foundation upon which a further comparison involving Mendix and OutSystems, on one hand, and on the other hand, other leading Low-Code platforms such as Microsoft Power Platforms, Appian, and ServiceNow is possible to be achieved. It would also extend the scope of this research and bring more comprehensive insight into the Low-Code ecosystem and its role in empowering businesses with flexible, scalable development solutions.

References

- [1] EZOFIS. (2023, September 11). The evolution of low-code platforms. <https://www.linkedin.com/pulse/evolution-low-code-platforms-ezofis/> (Accessed in July 2024)
- [2] NTT DATA. (n.d.). Low-code analysis. https://es.nttdata.com/documents/en_low_code.pdf (Accessed in July 2024)
- [3] Business Wire. (2019, October 29). IDC FutureScape outlines the impact "Digital Supremacy" will have on enterprise transformation and the IT industry. <https://www.businesswire.com/news/home/20191029005144/en/IDC-FutureScape-Outlines-the-Impact-Digital-Supremacy-Will-Have-on-Enterprise-Transformation-and-the-IT-Industry> (Accessed in July 2024)
- [4] Cyntexa. (2024, January 23). How low code or no code revolutionize the future of software development. <https://cyntexa.com/blog/low-code-no-code-the-future-of-softwaredevelopment/> (Accessed in July 2024)
- [5] Keller, D. (n.d.). OutSystems vs. Mendix: Clash of the low-code titans. <https://www.linkedin.com/pulse/outsystems-vs-mendix-clash-low-code-titans-dominik-keller/> (Accessed in July 2024)
- [6] Cyntexa. (2024, January 23). How low code or no code revolutionize the future of software development. <https://cyntexa.com/blog/low-code-no-code-the-future-of-softwaredevelopment/> (Accessed in July 2024)
- [7] Royce, W. (1970). Managing the development of large software systems. In Proceedings of IEEE WESCON (pp. 1–9). IEEE.
- [8] LinkedIn. (n.d.). Agile & Waterfall methodologies: How did Waterfall methodology emerge from the software engineering field? <https://www.linkedin.com/advice/3/how-did-waterfall-methodology-emerge> (Accessed in July 2024)
- [9] Borg, B. (2021, February 4). The history of how rapid application development became agile. <https://www.onpathtesting.com/blog/history-of-rapid-application-development-and-agile> (Accessed in August 2024)

- [10] Hohl, P., Klünder, J., van Bennekum, A., & Lockard, R. (2018). Back to the future: Origins and directions of the “Agile Manifesto” – Views of the originators. *Journal of Software Engineering Research and Development*, 6(1). <https://doi.org/10.1186/s40411-018-0059-z> (Accessed in August 2024)
- [11] Buchanan, I. (n.d.). History of DevOps: How development and operations teams came together to solve dysfunction in the industry. <https://www.atlassian.com/devops/what-is-devops/history-of-devops> (Accessed in August 2024)
- [12] Humble, J., & Farley, D. (2010). *Continuous delivery: Reliable software releases through build, test, and deployment automation*. Addison-Wesley Professional.
- [13] Project Management Institute. (2022, May 20). Waterfall methodology. <https://projectmanagement.ie/blog/waterfall-methodology/> (Accessed in August 2024)
- [14] Conrad, E., Misenar, S., & Feldman, J. (2010). Chapter 8 - Domain 8: Application development security. In *Eleventh hour CISSP: Study guide* (pp. 129-145). Syngress. <https://doi.org/10.1016/B978-1-59749-566-0.00008-4> (Accessed in August 2024)
- [15] Ewe Lutkevich, B., & Lewis, S. (n.d.). Definition: Waterfall model. <https://www.techtarget.com/searchsoftwarequality/definition/waterfall-model> (Accessed in August 2024)
- [16] Boehm, B. W. (1988). A spiral model of software development and enhancement. *Computer*, 21(5), 61–72. <https://doi.org/10.1109/2.59> (Accessed in August 2024)
- [17] Institute of Data. (2023, October 6). Understanding the Waterfall Model in software engineering. <https://www.institutedata.com/blog/waterfall-model-in-software-engineering/> (Accessed in August 2024)
- [18] Talreja, A. (2024, August 18). Understanding the Spiral Model in software development. <https://teachingagile.com/sdlc/models/spiral#planning-phase-identify-objectives> (Accessed in August 2024)
- [19] GeeksforGeeks. (2024, July 30). What is Spiral Model in software engineering? <https://www.geeksforgeeks.org/software-engineering-spiral-model/> (Accessed in August 2024)

- [20] Sarker, I. H., Faruque, F., Hossan, U., & Rahman, A. (2015). A survey of software development process models in software engineering. *International Journal of Software Engineering and Its Applications*, 9(11), 55-70. <https://doi.org/10.14257/ijseia.2015.9.11.05> (Accessed in August 2024)
- [21] Solinski, A., & Petersen, K. (2016). Prioritizing agile benefits and limitations in relation to practice usage. *Software Quality Journal*, 24(2), 447–482. <https://portal.research.lu.se/en/publications/prioritizing-agile-benefits-and-limitations-in-relation-to-practi> (Accessed in August 2024)
- [22] Codecademy. (2023, May 21). Rapid application development (RAD). <https://www.codecademy.com/resources/docs/general/software-development-life-cycle/rapid-application-development> (Accessed in August 2024)
- [23] Institute of Data. (2023, October 6). Understanding the Waterfall Model in software engineering. <https://www.institutedata.com/blog/waterfall-model-in-software-engineering/> (Accessed in August 2024)
- [24] Talreja, A. (2024, August 18). Understanding the Spiral Model in software development. <https://teachingagile.com/sdlc/models/spiral#planning-phase-identify-objectives> (Accessed in August 2024)
- [25] Coleman, G., & Verbruggen, R. (1998). A quality software process for rapid application development. *Software Quality Journal*, 7(2), 107-122. <https://doi.org/10.1023/A:1008856624790> (Accessed in August 2024)
- [26] Mishra, A. (2013). Suitability analysis of various software development life cycle models. *International Journal of Electronics Communication and Computer Engineering*, 98-101. https://www.researchgate.net/publication/286453113_Suitability_Analysis_of_Various_Software_Development_Life_Cycle_Models (Accessed in August 2024)
- [27] Svensson, R. B., Johansson, J., & Gall, D. (2013, July 2). Dynamic systems development method and rapid application development.
- [28] Voigt, B. J. J. (2004, January 20). Dynamic system development method. Zürich, Switzerland.
- [29] Sharma, L. (2021, July 7). DSDM: A step-by-step guide [2019]. ToolsQA. <https://www.toolsqa.com/agile/dsdm-guide/> (Accessed in August 2024)

- [30] Agile Business Consortium. (n.d.). Chapter 6: Process. <https://www.agilebusiness.org/dsdm-project-framework/process.html> (Accessed in August 2024)
- [31] Indeed Editorial Team. (2024, August 16). What Is the Dynamic Systems Development Method (DSDM)? <https://www.indeed.com/career-advice/career-development/rup> (Accessed in August 2024)
- [32] Satzinger, J. W., Jackson, R. B., & Burd, S. D. (2005). Object-oriented analysis & design with unified process. Boston: Thomson CourseTechnology. https://mygust.com/uploads/BOOK-Systems_analysis_and_design_in_a_changin.pdf (Accessed in August 2024)
- [33] Britton, C., & Doake, J. (2005). Chapter 1 - Introduction. In A Student Guide to Object-Oriented Development (pp. 1-19). <https://doi.org/10.1016/B978-075066123-2/50001-3> (Accessed in August 2024)
- [34] Hughes, R. (2016). Chapter 3 - Introduction to Alternative Iterative Methods. In R. Hughes (Ed.), Agile Data Warehousing for the Enterprise, A Guide for Solutions Architects and Project Leaders (pp. 31-54). <https://doi.org/10.1016/B978-0-12-396464-9.00003-5> (Accessed in August 2024)
- [35] Cho, J. (2009). A hybrid software development method for large-scale projects: Rational Unified Process with Scrum. Proceedings of the IIS 2009. Colorado State University-Pueblo. https://doi.org/10.48009/2_iis_2009_340-348 (Accessed in August 2024)
- [36] Indeed Editorial Team. (2024, August 16). RUP: Definition, phases, advantages and best practices. <https://www.indeed.com/career-advice/career-development/rup> (Accessed in September 2024)
- [37] Sacolick, I. (2022, April 8). A brief history of the agile methodology. InfoWorld. <https://www.infoworld.com/article/2334751/a-brief-history-of-the-agile-methodology.html> (Accessed in September 2024)
- [38] Highsmith, J. (2002). What is agile software development? Crosstalk: The Journal of Defense Software Engineering, 15(10), 4-9
- [39] A. Manifesto, "Manifesto for Agile software Development," 2001.

- [40] Hohl, P., Klünder, J., van Bennekum, A., et al. (2018). Back to the future: origins and directions of the “Agile Manifesto” – views of the originators. *Journal of Software Engineering Research and Development*, 6, Article 15. <https://doi.org/10.1186/s40411-018-0059-z> (Accessed in September 2024)
- [41] Pressman, R. S. (2010). Chapter 3. In *Software Engineering: A Practitioner’s Approach* (7th ed., pp. 69-93). McGraw-Hill.
- [42] Alvarez Duran, M. E. (2022, April 14). Your guide to understanding the Agile software development lifecycle. NAN-LABS. <https://www.nan-labs.com/blog/agile-software-development-lifecycle/> (Accessed in September 2024)
- [43] Toolagen. (2023, December 5). Understanding 6 crucial stages of the Agile software development process. LinkedIn. <https://www.linkedin.com/pulse/understanding-6-crucial-stages-agile-software-development-process-dwxtc/> (Accessed in September 2024)
- [44] Sekulic, N. (2023, April 28). 6 Stages of the Agile Software Development Lifecycle. GitKraken. <https://www.gitkraken.com/blog/6-stages-of-agile-development> (Accessed in September 2024)
- [45] Richardson, C., & Rymer, J. (2014, June 9). New development platforms emerge for customer-facing applications: Firms choose low-code alternatives for fast, continuous, and test-and-learn delivery. Forrester Research. <https://www.forrester.com/report/New-Development-Platforms-Emerge-For-CustomerFacing-Applications/RES113411> (Accessed in September 2024)
- [46] Mahlkow, M. (2023, October 27). The history of low-code. <https://fastgen.com/blog/the-history-of-low-code> (Accessed in September 2024)
- [47] Yong, Y. (2024, September 13). Software unicorn OutSystems sees “immense potential” in SE Asia as it taps opportunities for AI-powered low-code technology [Q&A]. TNGlobal Q&A and Interviews. <https://technode.global/2024/09/13/software-unicorn-outsystems-sees-immense-potential-in-se-asia-as-it-taps-opportunities-for-ai-powered-low-code-technology/> (Accessed in September 2024)
- [48] OutSystems. (2020). The state of application development: Is IT ready for disruption? Higher Education.

- [49] Sanchis R., Garcia Perales, O., Fraile F., Poler, (2019). Low code as enabler of digital transformation in manufacturing industry. *Applied Sciences*, 10(12), Article 10010012. <https://doi.org/10.3390/app10010012> (Accessed in September 2024)
- [50] Tisi, M., Mottu, J. M., Kolovos, D., De Lara, J., Guerra, E., Di Ruscio, D., Pierantonio, A., & Wimmer, M. (2019). Lowcomote: Training the next generation of experts in scalable low-code engineering platforms. <https://www.se.jku.at/lowcomote-training-the-next-generation-of-experts-in-scalable-low-code-engineering-platforms/> (Accessed in September 2024)
- [51] OutSystems. (n.d.). The transformative power of low-code apps. The Tech Hub. <https://www.outsystems.com/tech-hub/low-code/apps/#use-cases> (Accessed in September 2024)
- [52] Glib, O. (2023, May 16). 15+ low-code use cases: Apps you can build with low-code. Acropolium. <https://acropolium.com/blog/low-code-use-cases/> (Accessed in September 2024)
- [53] Mendix. Low-code examples and use cases. Mendix Resources. <https://www.mendix.com/low-code-guide/low-code-use-cases/> (Accessed in September 2024)
- [54] Bock, A. C., & Frank, U. (2021). Low-code platform. *Business & Information Systems Engineering*. <https://doi.org/10.1007/s12599-021-00726-8> (Accessed in September 2024)
- [55] Matvitskyy, O., Iijima, K., West, M., Davis, K., Jain, A., & Vincent, P. (2023, October 17). Magic Quadrant for enterprise low-code application platforms. Gartner. <https://www.gartner.com> (Accessed in September 2024)
- [56] Gartner. Positioning technology players within a specific market: Gartner Magic Quadrant. Gartner. <https://www.gartner.com/en/research/methodologies/magic-quadrants-research> (Accessed in September 2024)
- [57] Staniszewski, P. (2023, December 21). 2023 Gartner Magic Quadrant for Enterprise LCAP: Low-code keeps getting more relevant. Pretius. <https://pretius.com/blog/gartner-quadrant-low-code/> (Accessed in September 2024)
- [58] Vincent, P., Iijima, K., West, M., Matvitskyy, O., Davis, K., & Jain, A. (2023, October 18). Critical capabilities for enterprise low-code application platforms. Gartner. <https://www.gartner.com/en/documents/4853531> (Accessed in September 2024)

- [59] Butcher, M. (2018, June 5). Portugal's OutSystems raises \$360M from KKR and Goldman Sachs. TechCrunch. <https://techcrunch.com/2018/06/05/portugals-outsystems-raises-360m-from-krk-and-goldman-sachs/> (Accessed in September 2024)
- [60] OutSystems. Platform architecture overview. In OutSystems evaluation guide. <https://www.outsystems.com/evaluation-guide/architecture/> (Accessed in September 2024)
- [61] OutSystems. (2024, April 9). Setting up OutSystems. In OutSystems 11: Setup OutSystems infrastructure and platform. https://success.outsystems.com/documentation/11/setup_outsystems_infrastructure_and_platform/setting_up_outsystems/ (Accessed in September 2024)
- [62] Duro, J. (2021, February 24). OutSystems — Create a .NET extension — Date difference in months.ITNEXT. <https://itnext.io/outsystems-create-an-net-extension-date-difference-in-months-cedb4f5f4289> (Accessed in September 2024)
- [63] OutSystems, & Transfer Solutions. (2023). Cloud-native development report: The high cost of ownership. <https://www.outsystems.com/news/cloud-native-cost/> (Accessed in September 2024)
- [64] Battery Ventures. (2018). Mendix: Low-code application development platform. <https://www.battery.com/blog/mendix-case-study/> (Accessed in September 2024)
- [65] Mendix. Software development life cycle. Mendix Evaluation Guide. <https://www.mendix.com/evaluation-guide/app-lifecycle/> (Accessed in September 2024)
- [66] Mendix. Platform architecture. In Evaluation guide: Architecture. <https://www.mendix.com/evaluation-guide/enterprise-capabilities/architecture/platform-architecture/> (Accessed in September 2024)
- [67] Mendix. Cloud architecture. In Architecture. <https://www.mendix.com/evaluation-guide/enterprise-capabilities/architecture/cloud-architecture/> (Accessed in September 2024)
- [68] Keller, D. (2022, October 19). OutSystems vs. Mendix: Clash of the low-code titans (2023 update). Five. <https://five.co/blog/outsystems-vs-mendix-clash-of-the-low-code-titans/> (Accessed in September 2024)
- [69] Mendix. (n.d.). Open & extensible low-code development: Application extensibility - runtime. Mendix. <https://www.mendix.com/platform/openness-and-extensibility/> (Accessed in September 2024)

- [70] Auvee, E. (2024, May 12). Mendix vs. OutSystems: Comparison between the best two low code platforms. Impala Tech. <https://impalaintech.com/blog/mendix-vs-outsystems/> (Accessed in September 2024)
- [71] OutSystems. Developing with OutSystems: OutSystems evaluation guide. OutSystems. <https://www.outsystems.com/evaluation-guide/developing-with-outsystems/> (Accessed in September 2024)
- [72] Mendix. Experience the most extensive low-code IDE. Mendix. <https://www.mendix.com/platform/ide/> (Accessed in September 2024)
- [73] Mendix. (2024, October). Share the development database. Mendix. <https://docs.mendix.com/howto/data-models/sharing-the-development-database/> (Accessed in September 2024)
- [74] Mocke, R. (2022, December 12). Understanding entities and databases. Mendix Developer Blog. <https://www.mendix.com/blog/understanding-entities-and-databases/> (Accessed in September 2024)
- [75] OutSystems. Default platform server and database configurations. OutSystems. Retrieved from: https://success.outsystems.com/documentation/11/setup_outsystems_infrastructure_and_platform/setting_up_outsystems/default_platform_server_and_database_configurations/ (Accessed in September 2024)
- [76] OutSystems. (2024, August 29). Entity relationships. OutSystems 11 Documentation. https://success.outsystems.com/documentation/11/building_apps/data_management/data_modeling/entity_relationships/ (Accessed in September 2024)
- [77] OutSystems. Forge. OutSystems Community. <https://www.outsystems.com/forge/> (Accessed in September 2024)
- [78] Gadia, S. Mendix vs OutSystems: Faceoff between two best low code development platforms. Softude. <https://www.softude.com/blog/mendix-vs-outsystems-faceoff-between-two-best-low-code-development-platforms> (Accessed in September 2024)
- [79] Mendix. (2024, August). Error handling in microflows. Studio Pro 10 Guide. <https://docs.mendix.com/refguide/error-handling-in-microflows/> (Accessed in October 2024)

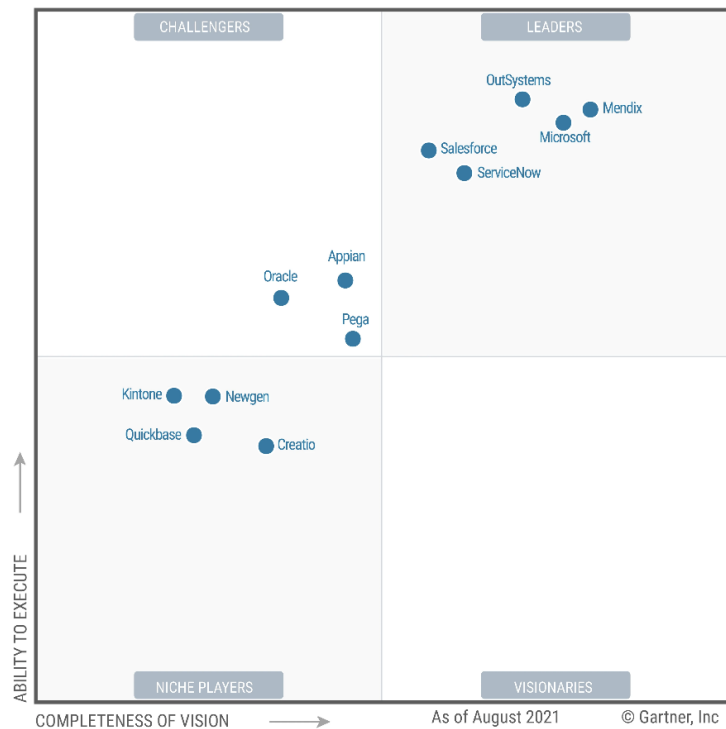
- [80] Goffe, B. (2021, August 6). Crushing technical debt with OutSystems, part 2: TrueChange. OutSystems.<https://www.outsystems.com/blog/posts/crushing-technical-debt-truechange/> (Accessed in October 2024)
- [81] Capitão, C. (2022, June 23). Handling with exceptions in OutSystems. <https://www.itup.io/blog-post/handling-with-exceptions-in-outsystems> (Accessed in October 2024)
- [82] Makadia, H. (2023, August). Mendix vs. OutSystems: A comprehensive guide to key differences. <https://marutitech.com/mendix-vs-outsystems/> (Accessed in October 2024)
- [83] OutSystems. Low-code integration with everything. <https://www.outsystems.com/low-code-platform/integration/> (Accessed in October 2024)
- [84] Amazon Web Services. What is REST? https://aws.amazon.com/what-is/restful-api/?nc1=h_ls (Accessed in October 2024)
- [85] OpenAI. OpenAI developer platform. <https://platform.openai.com/docs/overview> (Accessed in October 2024)
- [86] BidgeAll. (2024, August 20). Everything you need to know about OutSystems licensing. <https://www.bridgeall.com/2024/08/20/everything-you-need-to-know-about-outsystems-licensing/> (Accessed in October 2024)
- [87] OutSystems. OutSystems pricing. <https://www.outsystems.com/pricing-and-editions/> (Accessed in October 2024)
- [88] SaaSwothy. (2024, October). OutSystems pricing: Generate robust applications for your business in a hassle-free manner. <https://www.saaswothy.com/product/outsystems/pricing> (Accessed in October 2024)
- [89] Sri, A. (2024, June 7). Understanding the difference between ODC and OutSystems 11 (O11). <https://digitaltechworks.com/odc-vs-outsystems-11-key-differences-benefits> (Accessed in October 2024)
- [90] Djuriscic, S. OutSystems digital volunteers make the world a better place. <https://www.outsystems.com/blog/posts/app-dev-supporting-non-profits/> (Accessed October 2024)

- [91] Mendix. Mendix pricing - Flexible pricing for teams of all sizes.
<https://www.mendix.com/pricing/> (Accessed in October 2024)
- [92] SaaSworthy. Mendix: An all-in-one low-code platform to generate better software.
<https://www.saasworthy.com/product/mendix#pricing> (Accessed in October 2024)
- [93] Gasparyan, A. (2024, June 20). Mendix pricing: A short overview.
<https://devsdata.com/mendix-pricing/> (Accessed in October 2024)
- [94] Ang, A. Evolving with low-code and AI for agile innovation. Lancia Consult.
<https://www.lanciaconsult.com/insights/evolving-with-low-code-and-ai-for-agile-innovation-2> (Accessed in October 2024)
- [95] OutSystems. AI-assisted development: Definition, examples, and benefits.
<https://www.outsystems.com/tech-hub/ai-ml/ai-assisted-development/#benefits>
(Accessed in October 2024)
- [96] OutSystems. (2022, November 15). OutSystems introduces AI mentor system to accelerate developer productivity, dramatically improve code quality, and crush technical debt.
<https://www.outsystems.com/news/ai-mentor-system-introduced/> (Accessed in October 2024)
- [97] Vargues, P., & Rocha, N. (2023). To AI Mentor Studio and beyond. ONE Conference 2023.
<https://learn.outsystems.com/training/journeys/one-conference-2023-695>
(Accessed in October 2024)
- [98] OutSystems. (2024, May 8). Getting started with AI Mentor Studio.
https://success.outsystems.com/documentation/11/monitoring_and_troubleshooting_apps/manage_technical_debt/getting_started_with_ai_mentor_studio/
(Accessed in October 2024)
- [99] Mendix. Mendix AI-assisted development. <https://www.mendix.com/evaluation-guide/app-lifecycle/develop/ai/mendix-ai-assistance/> (Accessed in September 2024)
- [100] Mendix. (2024, August 21). Mendix AI Assistance (Maia). Studio Pro 10 Guide.
<https://docs.mendix.com/refguide/mendix-ai-assistance/> (Accessed in October 2024)

Appendix

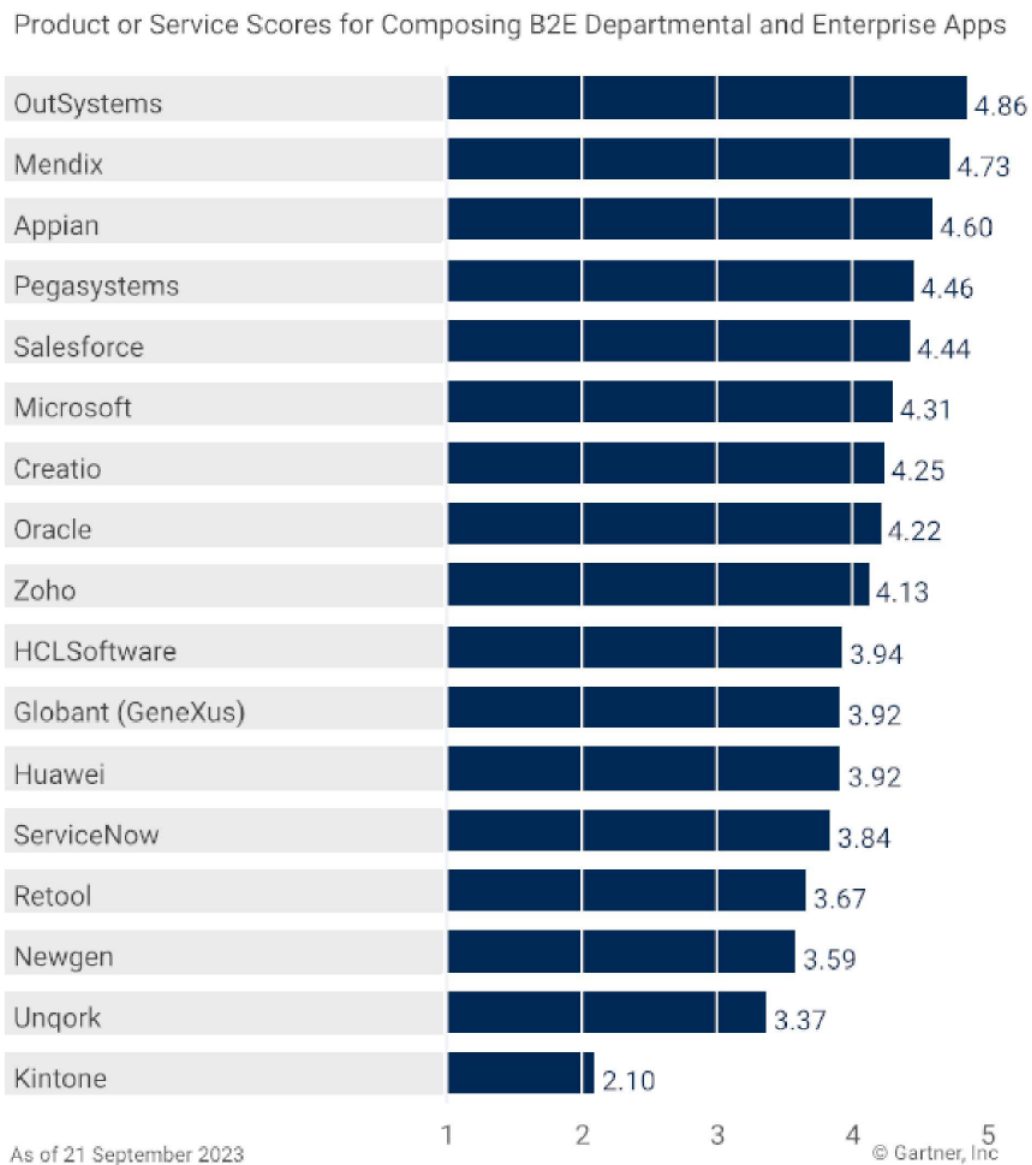
Appendix A

(1st Gartner’s 2023 and 2022 magic quadrant reports for Enterprise low-code Application platforms)



Appendix B

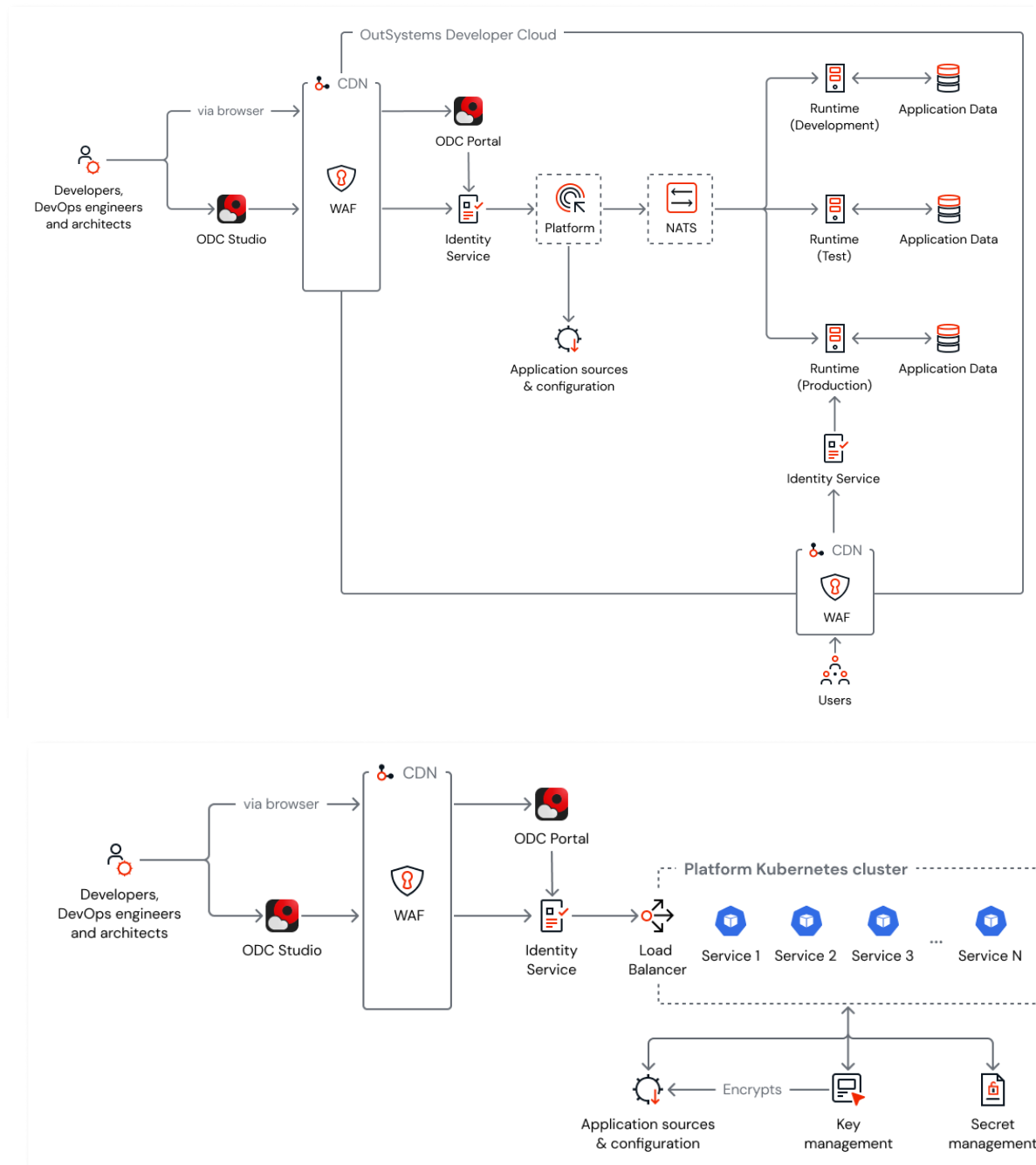
(1st Gartner's product or service scores for composing B2E Departmental and Enterprise Apps)



1st

Appendix C

(1st Cloud-native architecture diagram of OutSystems Developer Cloud – ODC)

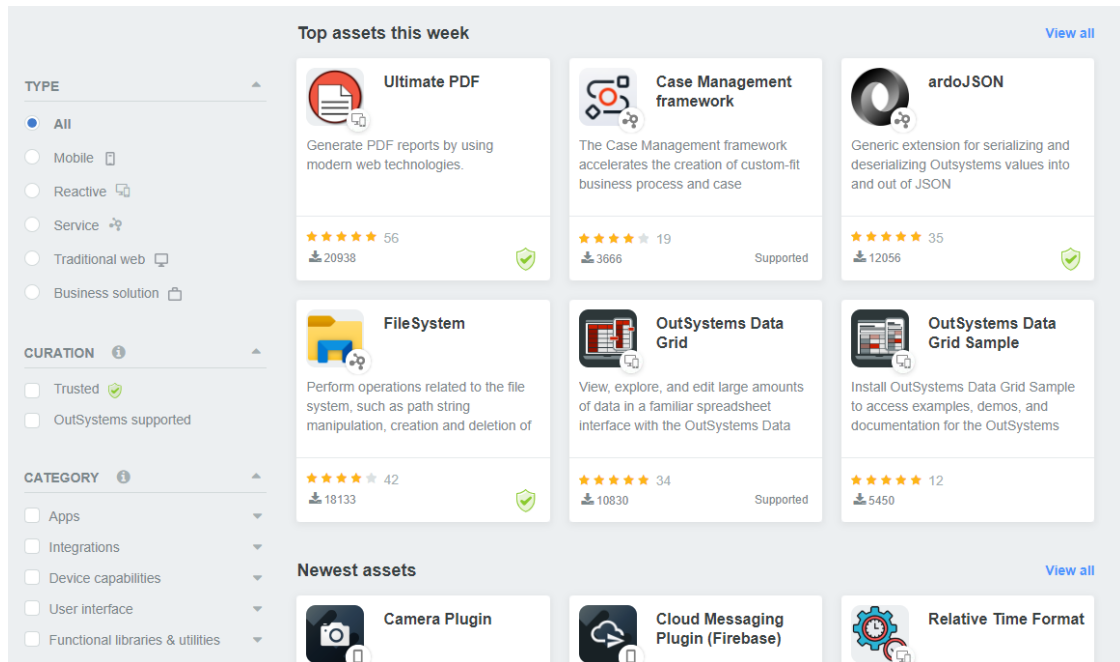


1st

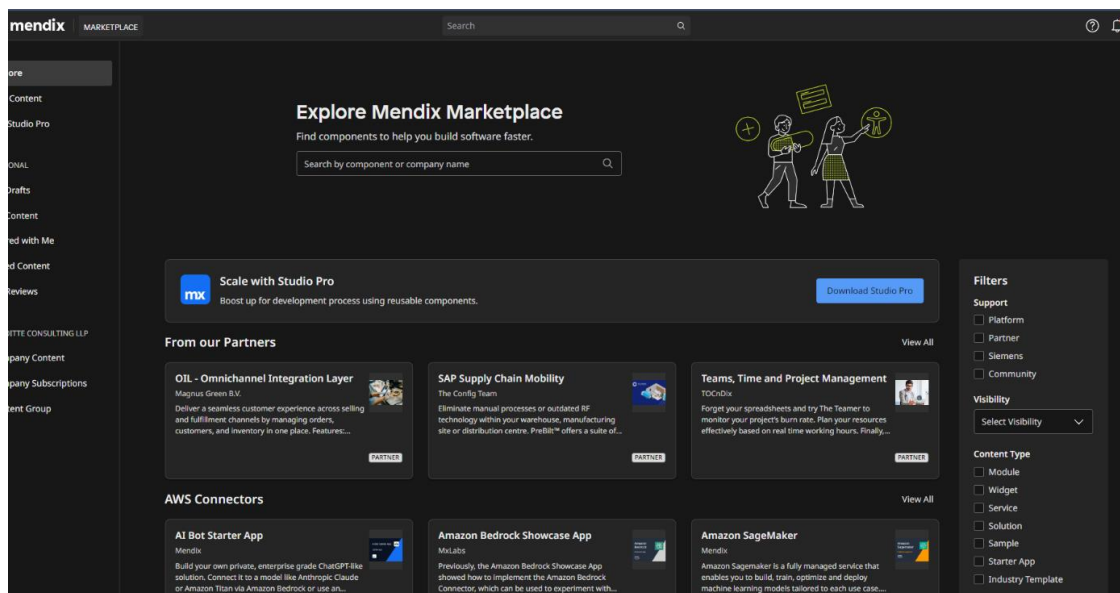
Appendix D

(1st Updated view of OutSystems Forge where the users can download components)

(2nd Mendix Marketplace where users can explore it and find components to help build software faster)



1st



2nd

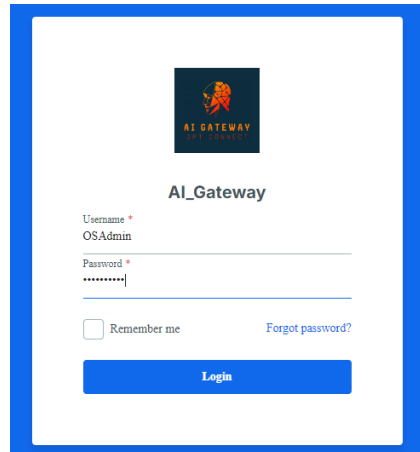
Appendix E

(1st Test-user Log-in screen)

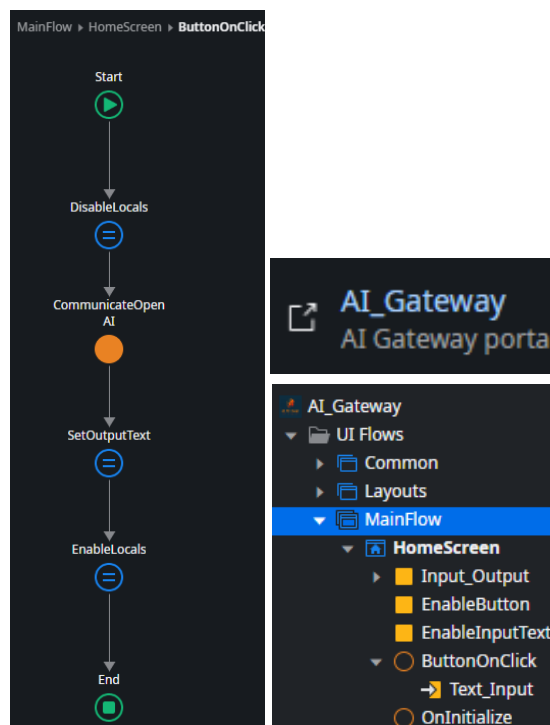
(2nd AI_Gateway action and main front-end module)

(3rd CommunicateOpenAI method and GPT_Connect_IS module)

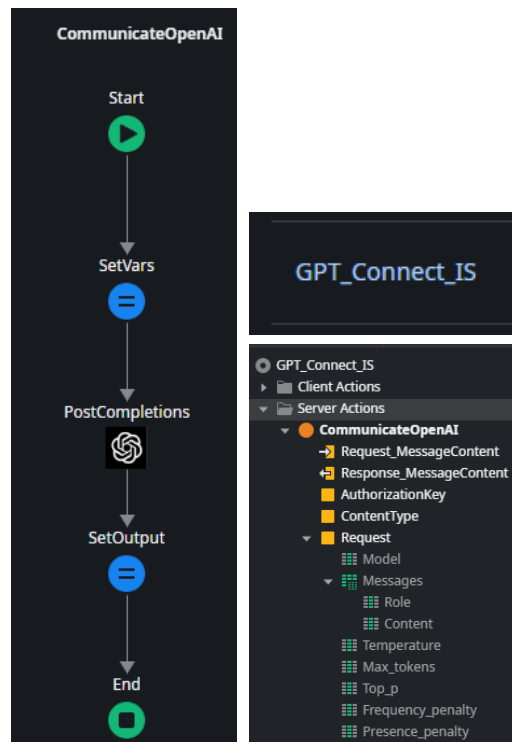
(4th AI Gateway Home screen with a self-made UX design)



1st



2nd



3rd

The screenshot shows the **AI_Gateway** interface with a header bar containing the logo and a **Test User** profile. The main heading is **Hello, ask a Question to OpenAI!**. Below this, there is a **Type Question:** section with a text input field containing the placeholder text "Please insert your question" and a blue submit button with a plus icon. Below the input field is an **AI Answer:** section with a large, empty white rectangular box for the response.

The screenshot shows the **AI_Gateway** interface after a question has been submitted. The header and heading are the same. In the **Type Question:** section, the text input field now contains the question "Hello, tell me about Low-code, what do you think of it?". In the **AI Answer:** section, the large white box contains a detailed paragraph of text explaining low-code development, its benefits (faster development, reduced costs, non-technical user involvement), and its limitations (lack of flexibility and customization compared to traditional coding).

4th

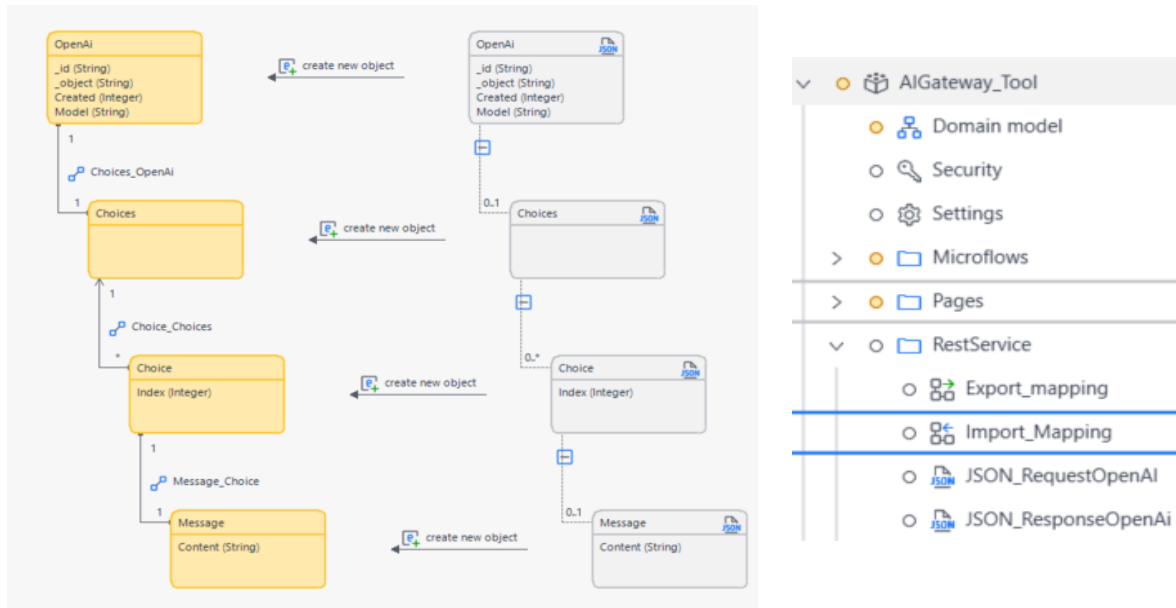
Appendix F

(1st refers to the Export Mapping)

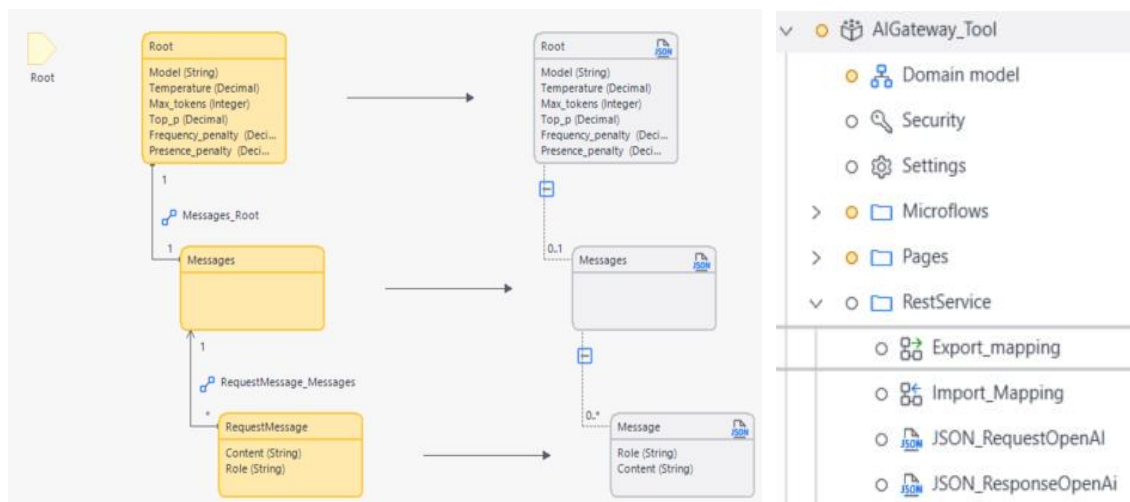
(2nd refers to the Import Mapping)

(3rd refers to the domain model used in the application)

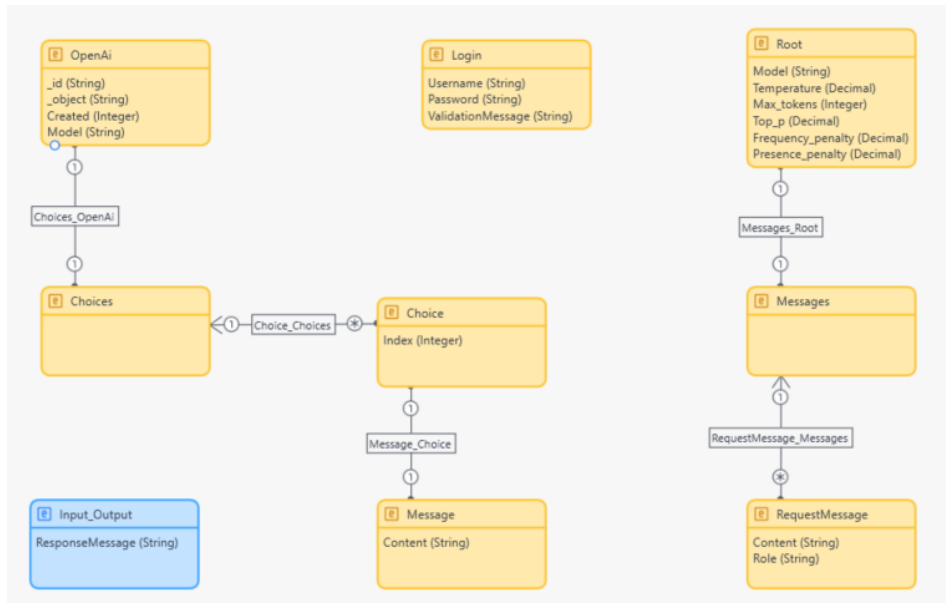
(4th refers to the application live demo)



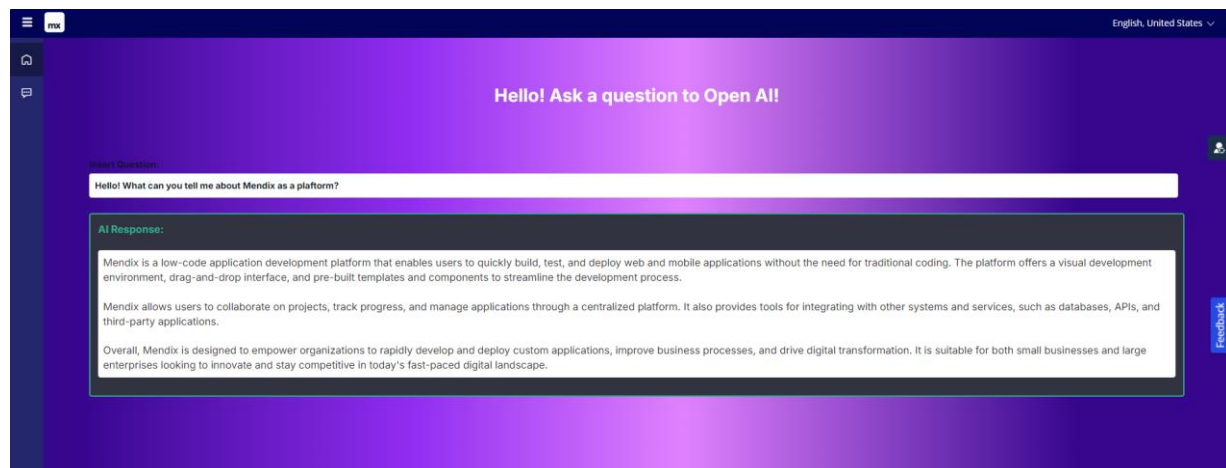
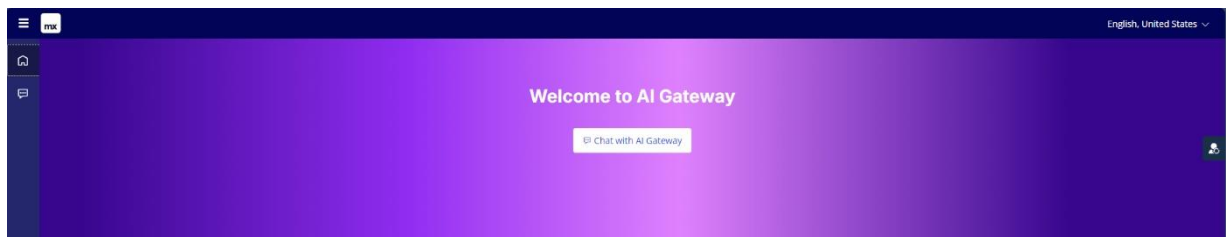
1st



2nd



3rd

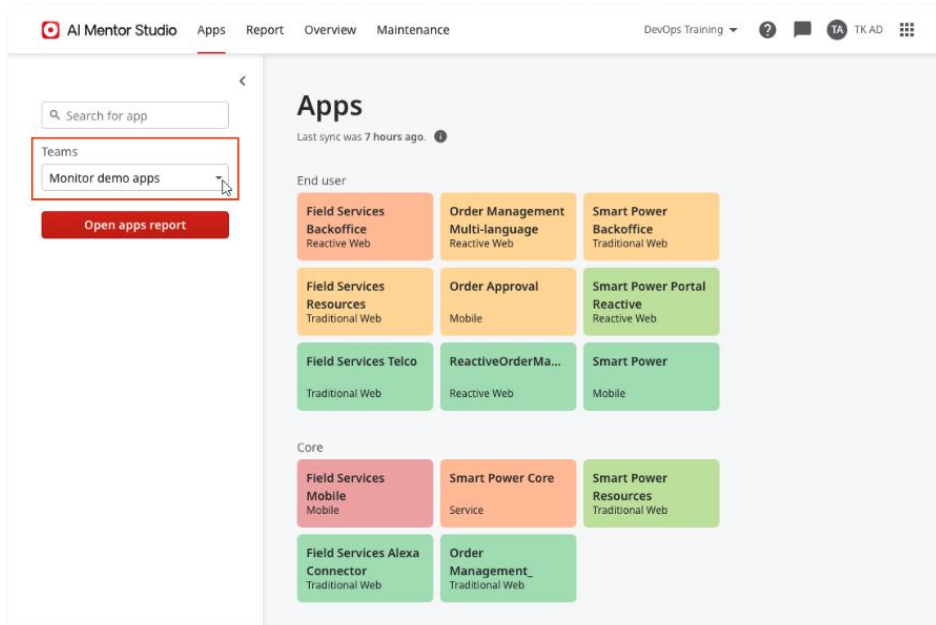


4th

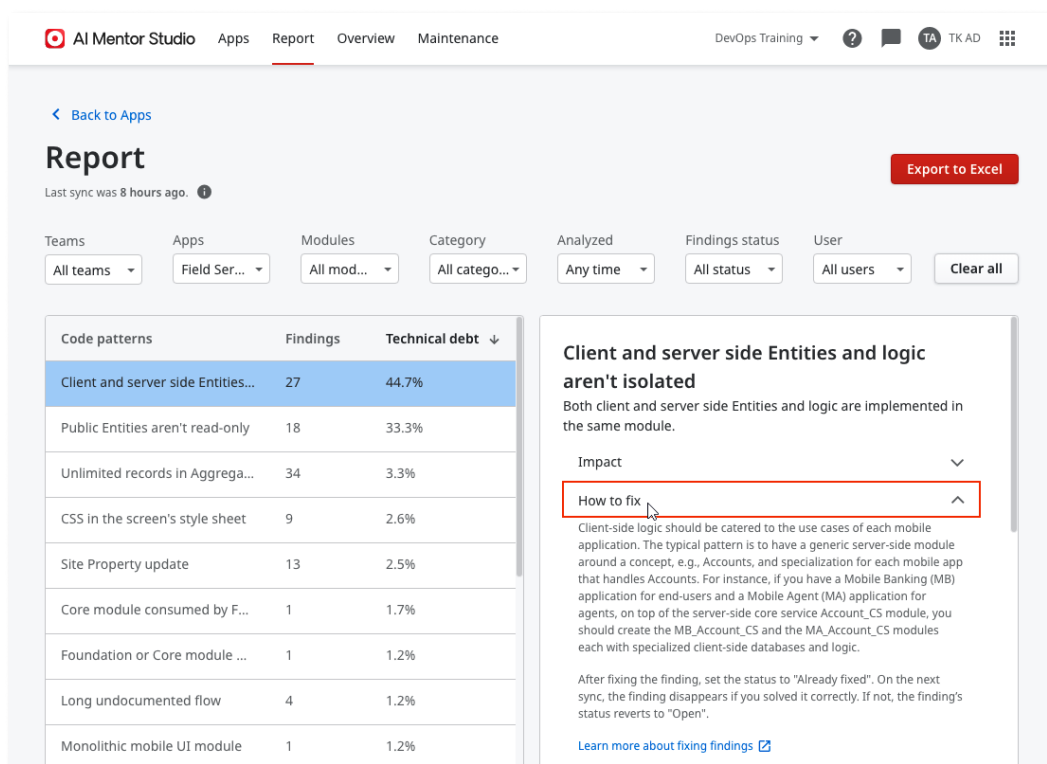
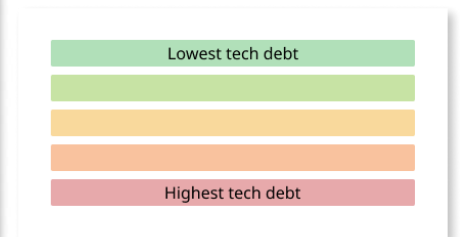
Appendix G

(1st refers to AI Mentor Studio and colour code that represent the technical debt)

(2nd refers to the report screen displayed after the user generated it)



1st



2nd

Appendix H

Article: AI-Driven Features in Low-Code Platforms: A comparative study of Mendix and OutSystems.

AI-Driven Features in Low-Code Platforms: A comparative analysis of Mendix, OutSystems and Microsoft Power Platforms.

Diogo Pina^{1,*}, Pedro Sebastião^{1,2}

¹ Department of Information Science and Technology, ISCTE-University Institute of Lisbon, 1649-026 Lisbon, Portugal; dfbpb@iscte-iul.pt (D.P.); pedro.sebastiao@iscte-iul.pt (P.S.)

² Instituto de Telecomunicações, 1049-001 Lisbon, Portugal

* Correspondence: dfbpb@iscte-iul.pt

Abstract: The adoption of Low-Code platforms has boosted enterprise application development, enabling a faster and more efficient delivery of valuable solutions. In the most recent years, Artificial Intelligence (AI) has become a key differentiator, by not only automating workflows, but also by improving error detection and so consequently enhancing overall productivity throughout the whole Information Technology sector. This paper aims to present a comparative study of AI-driven tools integrated into three leading low-code platforms: OutSystems, Mendix, and Microsoft Power Platform. OutSystems by featuring the AI Mentor System, which identifies security weaknesses, performance misbehavior, and overall coding errors, provides action points and useful guidance to developers. Mendix introduced Maia, an AI assistant offering intelligent recommendations and automation, such as an AI-assisted chat, a Best Practices recommender and also a Logic/Workflow Recommender, with the main goal of optimizing the development processes. Microsoft Power Platform also integrates its own tool, the AI Builder, a tool designed to empower users to create AI models for tasks such as form processing, object detection, and prediction with minimal technical knowledge.

This study intends to evaluate the tools based on their features, practical use cases, and impact on the software development lifecycle. By analyzing their contributions to key-metrics such as error-reduction ratio, intelligent automation, and developer productivity, finally this paper highlights how AI-driven tools empower organizations to deliver high-quality applications quickly and efficiently while optimizing its resources and work methods.

Keywords: Artificial Intelligence; Low-Code; OutSystems; Mendix; AI Mentor System; MAIA; AI Builder
