Degree Project in Computer Engineering
First cycle, 15 credits

Low-code Performance Evaluation and its Impact on Software Development

FREDRIK JANETZKY
MARKUS WALLIN

Stockholm, Sweden, 2024
Low-code Performance Evaluation and its Impact on Software Development

FREDRIK JANETZKY
MARKUS WALLIN

Degree Programme in Computer Engineering
Date: January 26, 2024

Supervisor: Siva Satya Sri Ganesh Seeram
Examiner: Ki Won Sung
School of Electrical Engineering and Computer Science
Swedish title: Low-code prestandautvärdering och dess påverkan på mjukvaruutveckling
Abstract

Low-code is a modern approach to software development and has grown rapidly in recent years. By hiding complexity behind layers of abstraction, it allows inexperienced developers to create complex applications in a shorter amount of time. This thesis sets out to evaluate the current state of low-code as well as a few of the available low-code development platforms and how well their finished applications perform.

A simple order management system was implemented in three low-code development platforms, Mendix, OutSystems, and Airtable, in order to gain experience in all of them. The platforms were then evaluated with respect to user experience and performance. The created applications load time and memory usage were measured in three different tests.

It was found that there are plenty of low-code platforms that exist for different purposes, and that they all have their own learning curve depending on how flexible they aim to be. The performance of their respectively created applications is also similar to each other, with the only major differences being by design.

Keywords

Low-code, Low-code development platform, Software development, Platform as a service, Citizen developer
Sammanfattning

Low-code är en modern metod för mjukvaruutveckling och har vuxit snabbt under de senaste åren. Genom att dölja komplexitet bakom lager av abstraktion, kan oerfarna utvecklare producera komplexa applikationer på kort tid. Detta examensarbetet utvärderar den aktuella statusen för low-code, samt några av de tillgängliga low-code plattformarna och hur deras skapade applikationer presterar.


Nyckelord

Low-code, Low-code development platform, Mjukvaruutveckling, Platform as a service, Citizen developer
Acknowledgments

We would like to thank Siva Satya Sri Ganesh Seeram for being our supervisor and guiding us through the process when conducting this thesis. We would also like to thank Ki Won Sung for being our examiner and for helping us complete our goals.

Stockholm, January 2024
Fredrik Janetzky        Markus Wallin
## Contents

1 Introduction  
1.1 Problem ........................................... 2  
1.2 Purpose ............................................ 2  
1.3 Goals ............................................... 3  
1.4 Research Methodology .......................... 4  
1.5 Delimitations ..................................... 4  
1.6 Benefits, Ethics and Sustainability ............. 5  
1.7 Structure of the thesis .......................... 5  

2 Background  
2.1 What exactly is low-code? ....................... 6  
   2.1.1 Rapid Application Development ............. 7  
   2.1.2 State of low-code ............................ 8  
   2.1.3 Benefits of low-code ....................... 9  
   2.1.4 Low-code concerns ......................... 10  
2.2 Related work .................................. 10  
2.3 Summary ......................................... 11  

3 Methods ............................................ 13  
3.1 Comprehensive review .......................... 13  
3.2 Implementation .................................. 14  
3.3 System Usability Scale ......................... 15  
3.4 Performance Data Collection ................... 17  

4 Implementation ................................ 18  
4.1 Platforms ....................................... 18  
   4.1.1 Mendix ....................................... 19  
   4.1.2 OutSystems .................................. 20  
   4.1.3 Airtable ..................................... 21
5 Result and analysis
  5.1 System Usability Scale .............................. 22
    5.1.1 Usability Evaluation .......................... 27
  5.2 Performance ........................................... 27
  5.3 Discussion ............................................. 30

6 Conclusion
  6.1 Conclusion ............................................ 31
  6.2 Limitations ............................................ 32
  6.3 Future Work ........................................... 33

References ................................................. 33
### List of acronyms and abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAGR</td>
<td>Compound annual growth rate</td>
</tr>
<tr>
<td>IT</td>
<td>Information technology</td>
</tr>
<tr>
<td>LabVIEW</td>
<td>Laboratory Virtual Instrument Engineering Workbench</td>
</tr>
<tr>
<td>LCDP</td>
<td>Low-code development platform</td>
</tr>
<tr>
<td>PaaS</td>
<td>Platform as a service</td>
</tr>
<tr>
<td>RAD</td>
<td>Rapid application development</td>
</tr>
<tr>
<td>SUS</td>
<td>System Usability Scale</td>
</tr>
<tr>
<td>UX</td>
<td>User experience</td>
</tr>
</tbody>
</table>
x | List of acronyms and abbreviations
Chapter 1

Introduction

Information technology (IT) is everywhere around us, and as our society progresses further, it forces people to get more and more in touch with it. The already large demand for developers or people with programming knowledge has increased considerably throughout the years and is not expected to slow down.

Despite the rapid development of society and the integration of programming into the curriculum at an early age, there remains a shortage of individuals with adequate education and expertise in the field. The scarcity of developers is one of the key factors that made low-code emerge.

Low-code is a modern approach to software development that offers a more efficient and flexible development process compared to traditional practices. Allowing a larger number of users to develop applications, regardless of whether they possess programming knowledge or not.

Ever since the term "low-code" was coined by Forrester in 2014 [1], the market for Low-code development platforms (LCDPs) has been growing rapidly, year after year. The rise of low-code development has really challenged the ways of traditional software development methods. Many large businesses and organizations have recognized this and integrated low-code solutions into their workflows or even gone as far as to develop their own platforms. The adaptation of LCDPs allows them to develop applications rapidly, efficiently, and cost-effectively, considering they do not have to hire developers with extensive experience or programming knowledge.

Today, LCDPs are being utilized by the majority of enterprises [2] to build both internal solutions for management and productivity, and for products to meet their customers needs. What kind of impact will low-code have on IT as a whole? Low-code has already left a noticeable imprint on the industry.
However, is this approach to software development beneficial for the future, or is it by any chance decreasing innovation within the industry? Will it enhance the evolution of coding or, in fact, slow it down? How will it affect the current and new developers?

This thesis will answer some questions regarding low-code, including its benefits as well as its limitations. Through an examination of the aforementioned questions, the authors of this thesis intend to provide a comprehensive understanding of low-code. Additionally, this research attempts to determine whether low-code is a lasting phenomenon or merely a passing trend.

1.1 Problem

The usage of low-code and LCDPs has exhibited a rapid surge in recent years. However, it is essential to investigate what differences there are between different platforms and whether this upward trend is expected to persist. Could low-code development potentially supplant conventional programming in the future, and what would the resultant implications be? Is it possible that a shift to low-code leads to alternations in productivity levels, methodologies of software development, innovative practices, and the demand for software developers?

1.2 Purpose

The purpose of this thesis is to answer some important questions that come to mind when reflecting on the evolution and integration of low-code. The results of the thesis may also be of interest to businesses looking into low-code solutions for their needs. Even though low-code has the potential to improve the development efficiency of new applications and increase the speed at which they the market, low-code development may become such a capable utility that it drastically reduces the need for multiple software developers in the future.

From a business point of view, this thesis aims to answer some questions regarding whether there is an economic advantage to developing their own traditional application from the ground up or if it is more cost-effective to leverage the benefits of low-code platforms to suit their needs. Delegating development tasks to people with basic or no programming skills. Will that affect the contact between developer and customer, and if so, in what ways? These questions will also give necessary answers regarding low-code from an
engineer’s point of view. Finally, the purpose of the thesis is also to allow the authors to learn and contribute in a scientific manner. Through researching related topics, testing, and finding answers to the aforementioned questions, the authors will accomplish this.

1.3 Goals

The main goal of this thesis is to determine whether low-code is a lasting phenomenon or merely a passing trend. What impact could it have on society from a developer’s point of view? To accomplish this goal, which defines the entire thesis, it has to be broken down into smaller goals. This will help the main goals being carried out.

1. One of the first goals is to find out more about low-code and its possibilities. By doing this, the authors of this thesis will get a general understanding of what the current low-code landscape looks like and how they may develop to improve its current state.

2. Is low-code here to stay, or is it merely a trend? By testing out different low-code platforms, the authors of this thesis will gain some knowledge that will make it possible to compare it to traditional coding based on prior knowledge.

3. What differentiates the different low-code platforms? By experiencing and testing platforms, knowledge will be gained about the differences, both in terms of usability and performance.

It is also necessary to say that one of the goals of this thesis is to meet the requirements for a bachelor’s degree. To achieve a bachelor’s degree, a student has to work and finish a bachelor’s thesis [3]. By doing this thesis, the student should be able to do the following things.

- With a holistic view, independently and creatively identify, formulate, and manage issues and analyze and evaluate various technical solutions.

- Plan and with adequate methods carry out tasks within given frameworks.

- Design and manage products, processes, and systems.

- Orally and in writing give an account of and discuss information, problems, and solutions in dialogue with different groups.
In the author’s point of view, these goals are very important to show with the aim of being able to work independently as an engineer.

1.4 Research Methodology

To answer these questions, a qualitative research methodology is carried out. With the aim of gaining a deep understanding and knowledge of the current status of low-code and low-code platforms, as well as better understand its growth and what can be expected from it in the coming years. It was chosen to do qualitative research over quantitative because there are a smaller number of questions that are going to be in focus. In order to gain that knowledge, various methods were selected.

A large part of this study is to research the topic extensively and look at various reports from different outlets. This is to gather general information about low-code. However, it is also to get a grasp of what opinions others have on the topic so that they can be compared to the findings. Related work from other authors will also be looked at to contrast their results and use that information to strengthen the findings found in this thesis.

Firstly, it is important to get hands-on experience with low-code platforms in order to better understand how they work and what can be achieved by using them. Since the authors are both computer science students with various experience in many commonplace programming languages and having completed many smaller projects and programs over the years, it will give them a good starting point when comparing traditional programming languages to low-code development platforms. Comparing low-code to traditional code will give them an insight into how difficult low-code is to use, how complex the code and solutions can be, and most importantly, how time-efficient it is.

Other than getting some experience with low-code platforms, a series of tests will be performed on the low-code-developed applications to test their efficiency. By doing this, it will be shown which platform is most suitable for specific tasks.

1.5 Delimitations

Considering the scale of this thesis is at a bachelor level, certain time and resource limitations decrease the scope of the project. Seeing as there are over 200 platforms [4] currently available on the market, examining more than a few platforms would take a considerable amount of time and is not
feasible. Furthermore, some established platforms charge a cost to use their full experiences, and combined with the lack of resources, only free or trial versions of platforms will be used. However, most paid platforms offer trial versions with limitations, allowing most functionality to be tested free of charge.

1.6 Benefits, Ethics and Sustainability

The results of this thesis aim to provide a deeper understanding of low-code. Companies are spending vast amounts of money, resources, and time when it comes to developing new systems and applications. This project provides some insight into whether it is beneficial to use LCDPs in regard to economic favor and time consumption compared to hiring a traditional developer. Thus, it is of great interest and beneficial for companies. However, there is also an interest for the individual developer who aims to use a low-code platform for future work, considering this project aims to explain the benefits of low-code platforms and what they are most suitable for.

In terms of ethics, the result of this project might be of great favor for LCDPs, which may harm the reasons for hiring an experienced developer. This would potentially decrease the number of job opportunities, assuming the choice of a LCDP has too many favorable advantages. However, there is also the concern of being locked to a development platform and what could happen if a platform ceases to operate. This possibility may dissuade developers from using LCDP.

For sustainability, if low-code is a more efficient alternative to software development, it could also mean that less energy and resources are necessary to create the same product compared to conventional coding.

1.7 Structure of the thesis

Chapter 2 presents the history and current state of low-code, as well as what makes it attractive to developers. Chapter 3 presents the methodology and the four methods used in the project to solve the problems stated. Chapter 4 covers the selection of low-code platforms and the implementation of the application to be used for testing. Chapter 5 presents the result from the testing and how it has been evaluated. Chapter 6 is a conclusion of the finished work and discussion about what future works could cover.
Chapter 2

Background

This chapter aims to provide the reader with sufficient background information about low-code. Going over its brief but impactful history as well as how it is currently being used by businesses and developers all over the world.

2.1 What exactly is low-code?

Low-code has been around for quite a while now, even though it has not exploded in popularity until the last couple of years. Low-code development can be traced back to the early 2000s or even the late 1990s, when there was an interest in making programming more accessible for non-technical users. It was in this era that the definition of visual programming languages appeared.

These early visual languages were graphical representations of different programming concepts. Most of these languages used block diagrams and flowcharts. This helped the users make applications without having the knowledge to write traditional code.

One of the first examples of visual programming languages was Laboratory Virtual Instrument Engineering Workbench (LabVIEW). LabVIEW was first launched in 1986 by National Instruments. It is still in use, and their latest version was released in Q1 2023. LabVIEW is designed to enable engineers and scientists to facilitate custom automated measurements and automated systems using a graphical programming language that represented code as visual blocks connected by wire. This was one of the founding platforms that used one sort of visual programming that set the standard for future development in the area [5]. LabVIEW was inspiring new visual programming platforms to emerge, which offered a range of pre-built components and a lot of different visual development tools. This made the expression Rapid
2.1.1 Rapid Application Development

RAD is a software development methodology that really enhances the delivery of different applications by emphasizing quick and iterative development cycles. This means that the development is focusing more on prototyping with quick feedback from the different cycles at the expense of less planning. This will accelerate development processes; however, it may result in more bugs that otherwise would have been caught. This problem is erased through the use of multiple iterations. This will help developers create applications that suit the end user's requirements early in the project stages. There are a couple of characteristics that RAD stands for.

- Iterative development is one of the main advantages of RAD. By dividing the process into smaller, more manageable iterations, completions of smaller or more specific functionality or features can be made.

- Prototyping is the second largest advantage. This helps the process gather feedback from the different iterations by finishing prototypes early in the process. These prototypes are reviewed by the stakeholders. In some cases, if a developer or stakeholder is not satisfied with a feature, they can redefine it early in this phase.

- By setting strict time limits for RAD projects, the process will be more efficient. It will allow the development to progress at a quick pace. It will also help the developers to see the results at an earlier stage compared to a normal project, which will also decrease the time-to-market.

- By reusing different modules and components whenever it is possible, a lot of time and effort can be saved. This is known as reusability. Instead of developing similar or the same components from scratch, an already existing functionality can be used again.

RAD is the baseline of all the low-code platforms [6]. Even though RAD is a unique model for software development, it does not have to be better in all cases in comparison to more traditional methods such as the "waterfall model." The waterfall model is one of the most traditional models when it comes to engineering projects; it is also referred to as a linear-sequential life cycle.
model. This thesis will not thoroughly explain it. However, a brief description of it is needed to see the major characteristics and disparities between the two development methodologies [7]. Just as the name explains, the waterfall model is a breakdown of project phases in a sequential order where the output of one phase acts as the input for the next phase. There are usually six different sequential phases in the waterfall model.

- Requirement Gathering and analysis.
- System Design.
- Implementation.
- Integration and Testing.
- Deployment of system.
- Maintenance.

The advantages of the waterfall method are, in general, that a plan for the entire project is planned in the beginning, which makes all the issues easier to identify early. Constructing a plan for the entire project simplifies the understanding of the project. This makes it easy to follow the arranged tasks. It also encourages large teams to work towards a common goal. From a developer’s point of view, it also enhances good coding habits since it is usually defined before the actual implementation begins [8].

The waterfall methodology is currently prevalent and widely adopted. However, it is not universally suitable for all scenarios due to its inherent limitations. In such contexts, RAD could offer a more advantageous alternative. Waterfall is not very adaptive; when a fatal bug or flaw is found, entire processes risk having to be rewritten from the ground up. The waterfall method usually does not involve the end user or client until the development process is complete, compared to RAD where feedback is given after almost every iteration. Projects arranged with the waterfall method typically take longer. The average time for developing a software is 4-5 months according to Victoria Puzhevich [9].

### 2.1.2 State of low-code

Today, low-code is widely deployed among both large enterprises and small businesses. According to Forrester, 84% of enterprises have adopted low-code solutions to reduce strain on resources and deliver products faster [2].
The market size for low-code is reportedly worth somewhere between 7.61 billion and 16.3 billion USD, with a Compound annual growth rate (CAGR) of 27.8-29.8% [10][11]. While these statistics do not necessarily reflect the true capabilities of low-code, they demonstrate the substantial footprint that low-code has already established within the industry, solidifying its status as a proven tool for developers.

The market offers a wide array of platforms catering to diverse demands and requirements, with more than 200 platforms present, and the list is growing [4]. Notable conglomerates like Microsoft and Google have entered the market, offering their proprietary solutions. Most LCDPs operate as a Platform as a service (PaaS), indicating that their platform is cloud-based.

### 2.1.3 Benefits of low-code

Low-code solutions offer several compelling reasons to be considered in the development process. The following factors make low-code an appealing choice for businesses of all sizes:

- **Efficient and fast development:** Low-code allows developers to efficiently develop simple or complex programs. According to a study by 451 Research [12], using low-code compared to a coding language can potentially reduce development time by 50-90%.

- **Accessibility for non-programmers:** This allows people without a programming background to develop applications. These developers are often referred to as "Citizen Developers" [13] within the low-code domain.

- **Cost-effectiveness:** Low-code reduces the overall cost of application development due to its efficiency and user-friendly nature. Instead of hiring large teams of skilled developers, smaller teams can accomplish similar results within the same time frame, allowing businesses to save resources.

- **Flexible development:** Low code can be used to quickly prototype ideas and provide ease of modification. This can be helpful during development to better reach the original vision of the application.

There are more reasons as to why one would consider using low-code applications for their project, such as integration capabilities with other services and maintenance; however, the most lucrative reason is the fast development and the potential cost savings that it may entail.
2.1.4 Low-code concerns

While there are plenty of benefits, there are also plenty of concerns when it comes to low code. Some of the major concerns are:

- **Customization restrictions**: The power of low-code is that you can quickly develop programs using pre-defined code; however, this also means that you sacrifice some control over the code. Given a sufficient level of expertise and time allocation, traditional coding exhibits boundless potential. Conversely, the utilization of low-code constrains developers to the functionalities sanctioned by the specific low-code platform.

- **Application Performance**: Without the ability to fine-tune and change specific code snippets, the performance may not reach its full potential as would otherwise have been possible with a traditional language.

- **Security**: Typically, you have to rely on the platform to ensure that the security that is offered is sufficient.

- **Vendor lock-in**: Applications developed within the confines of a LCDP tend to possess proprietary attributes, rendering the migration of low-code-built applications challenging, if not completely impossible. This can become a costly obstacle in case the platform turns out to be too limited or, worse, ceases to operate.

In some cases, low-code’s strengths can also be its weaknesses. Sacrificing level of control for development efficiency comes with plenty of pros and cons; hence, it is detrimental to do proper research on LCDPs before committing to using one for a project. It is also crucial that the companies developing LCDPs keep their platform up to date and strive to reduce limitations while maintaining ease of use in their platform.

2.2 Related work

A sizable comparison and taxonomy across multiple LCDPs has been conducted by A. Sahay et al. [14]. This report examined eight platforms, aiming to differentiate their capabilities and assess the functionalities they offer. While these platforms often accomplish the same end goal, they have different features contributing to that goal. The findings from this report shed light on the disparities among evaluated platforms, providing valuable insights
that will inform the selection process of platforms to be explored in our own research.

2.3 Summary

In this chapter, the origins and concepts of LCDPs were explored. Overall, low-code development has become a proven tool for developers; however, it is still growing in popularity, mainly because it is easy to learn and use. For a developer today, there are many different platforms on the market to choose from. At a glance, the available platforms all offer flexibility, cost-effectiveness, and easy maneuverability, even for non-programmers. However, there are some concerns regarding performance and customization options, especially for large-scale projects.

Briefly, low-code can be described as a modern approach that attempts to accelerate and simplify software development. By aiding the developer with a visual interface and drag-and-drop components, it allows experienced and inexperienced developers alike to create complex products and solutions.
Background
Chapter 3

Methods

In this chapter, the methods used to create and evaluate the experiment are explained. To achieve the goal of answering our research questions, the project was organized into four distinct phases. The initial phase involves conducting comprehensive research on the subject and making a plan for the upcoming experiment in Section 3.1. The second phase involves the development of the application on various LCDPs which is explained in Section 3.2. The third phase encompasses the data collection from the platforms where the application is developed, with a focus on user experience in Section 3.3 and performance in Section 3.4. Lastly, the fourth phase is dedicated to the evaluation of the collected measurements.

3.1 Comprehensive review

The project consisted of four different phases. The first phase that was conducted included a comprehensive review of the literature regarding the subject. A systematic review was first considered. However, due to the limitations of time, a fully systematic approach would not be the best choice in this case. In a paper conducted by Maria J. Grant and Andrew Booth, it is written that a systematic review would consist of in-depth research and would retrieve all relevant information and articles within the area. This would put the authors of this report outside the limits of time for this report. A comprehensive review, however, can be chosen to be an in-depth search for information, although with a more narrow aspect. This would still give great accuracy, even though some relevant information was lost [15]. Figure 3.1 shows how the process is made with the literature study and the following test with the thesis writing.
3.2 Implementation

In order to conduct an evaluation of various LCDPs, one of the first steps that was of vital importance was to design an application scenario that aligned with the project’s objective. When designing the application, two measurements would be taken into consideration. First of all, our primary objective was to use a conventional System Usability Scale (SUS) methodology to measure and assess the User experience (UX) among the different LCDPs. This would give us a chance to evaluate the different functionalities but also to identify some inherent limitations across the platforms.

The second way of evaluating the platforms involved measuring their performance. This is explained in Section 3.4. While developing the test application, some constraints were encountered. The most distinguished one was the limitations in time. The limited time frame compelled us to refrain from developing a complex application. Our goal was also to develop the app on multiple LCDPs. A temporal time allocation was needed for each time the application was built on a new LCDPs considering they had different operational paradigms.

We decided to develop an order management system for pre-existing enterprises designed to facilitate the placement and allocation of orders among various staff members. Everything is overseen centrally by the administrator.

The development of this application needed some specific features, strategically selected to assess the constraints and capabilities among the various LCDPs.

Among these features were actions such as adding and removing orders,
assigning specific employees, and user authentication by adding login and logout procedures. The initial design of this application is illustrated in Figure 3.2.

![Figure 3.2: Overall View of the application](image)

### 3.3 System Usability Scale

To evaluate the user experience among the different low-code platforms, we used SUS. It is a fast and reliable tool for measuring usability in products and services. SUS consists of a ten-item questionnaire, where the answers are graded on a five-point scale. Where a five equates to "strongly agree" and a one means "strongly disagree." The system was created by John Brooke in 1986 but is still in use when evaluating a wide range of different services, such as websites, software, hardware, and applications. The questions are divided into odd and even ones. Where the odd questions have a positive emphasis and the even questions a negative one. This alternation of positive and negative questions is to prevent response bias [16].
Once the grading is complete, the total score can be calculated. The calculation is performed by adding up the scores from all questions, where each question’s score will range from 0 to 4. For odd questions (1, 3, 5, 7, and 9), the score for each question is calculated by taking the scale position minus 1. For even questions (2, 4, 6, 8, and 10), the score is calculated by taking 5 minus the scale position. After grading each question, all question’s score are summed together and multiplied by 2.5, giving us a final score between 0 and 100. One thing worth noting is that even though the final score ranges from 0 to 100, it is not to be confused with percentages. It should only be considered a rank in terms of percentile ranking. Also, the reason why the calculation is different for the odd and even questions is because the odd questions generally generate a more positive response, while the even ones generate a more negative response. That is why the scores for the even ones have to be inverted.

A higher score indicates better usability [16]. SUS was developed to provide a point estimate measure of usability and satisfaction. However, SUS has many benefits beyond these that make it suitable for our project. It made it easier for us to compare different tasks within the same interface, but is also provided a way of measuring the different platforms among themselves. What an acceptable score is has been discussed among different authors. In a study by Aaron Bangor, Philip T. Kortum, and James T. Miller, where more than 10 years worth of SUS data was collected, there is the conclusion that the final result should be at least 70. If you were in the upper 70s or higher, the result was defined as good. If the result was below 70 or in the 1st quartile, the result was poor and could use improvements [17]. As seen in Figure 3.3 the final score is divided into four different quartiles, where it is preferable to be in the top quartiles [17].

Figure 3.3: System Usability Scale scores where the quartile and acceptable SUS score is included[16].
3.4 Performance Data Collection

In order to sample the performance of the web applications created using the selected LCDPs, the application specified in Section 3.2 was implemented on every platform. Once they were created, they were filled with the same set of data. To measure the performance, Chromium was selected as the browser to conduct the tests on. Using the runtime performance analyzer found in the Chrome DevTools, the load time for each web application was measured using 4x CPU slowdown. The memory usage for each application was also measured using heap snapshots found in the same toolkit. Three different tests were performed for each web application, and each test was performed three times to get an average value and reduce the risk of anomalies. The different tests were as follows:

- Test 1 consists of measuring the load time of the application’s HomePage.

- Test 2 consisted of measuring a regular use case of the application. The route for this test started at HomePage. A new order was created. The new order was then marked as completed, and lastly, we navigated to the completed orders page to view the completed order.

- Test 3 was performed using a separately created web application. This application consisted of a large list containing a total of 1000 elements in order to simulate a heavier load. Performance was then measured based on the time it took to load the page.
Chapter 4

Implementation

This chapter covers the criteria and selection of platforms that were tested. Every platform is briefly introduced, and the test application that was developed using the platforms is shown.

4.1 Platforms

As previously mentioned, our selection of LCDPs was constrained by a couple of considerations; the biggest one was the limitation in time. This simply concluded that we did not have the time to explore all of the available platforms that are on the market today. This made us have a more narrow and focused approach to the platforms that we chose. To accomplish this, we made a set of criteria that needed to be fulfilled to choose a specific platform.

- The first and main criterion was free of charge. We prioritized LCDPs that offered free development capabilities.

- The second criterion we had was that we wanted LCDPs with a proven market presence. The reason for this was that it indicated that the platforms had widespread use and were proven to work in a real-world scenario with a certain effectiveness.

- The last criterion was that we sought to find platforms that offered strong flexibility in customization, considering we wanted to test the platforms thoroughly.
4.1.1 Mendix

Mendix was selected as the first platform. It is an all-purpose LCDP founded in 2005 as a means to bridge the business-IT gap and quickly developed to be one of the leading actors among LCDPs. Mendix is an attractive choice for both non-technical users and developers because of its user-friendly interface that allows for short development cycles, making it easier for developers to see the design choices they make. According to the Forrester Wave, Mendix boasts many top features that are suitable for simple apps but also for more extensive ones [18]. Mendix offers a free version for non-commercial use meaning that all of the criteria was met and allowed us to test the platform. As shown in Figure 4.1 is the final application done in the Mendix environment.

![Mendix Application](image)

Figure 4.1: View of the application developed in Mendix platform
4.1.2 OutSystems

The second selected platform was OutSystems. Outsystems is another all-purpose platform and is also in a position where it is a direct competitor to Mendix. Both platforms offer similar features and capabilities. This meant that a comparative analysis between the platforms could be done when examining their respective functionality. Figure 4.2 shows that the application had a similar look as the one done in Mendix.

![View of the application developed in OutSystems platform](image)

In contrast, Mendix had a more automated approach in this regard. Consequently, OutSystems appeared to be better suited for users with a greater level of expertise and experience in application development.
4.1.3 Airtable

The selection of the third platform faced some challenges, primarily related to the first criterion. While a number of LCDPs offer free versions of their platforms, they are often quite restrictive and limited. However, by narrowing our research, we ended up selecting Airtable as our final platform. Airtable represents a cloud-based LCDP that focuses on a user-friendly spreadsheet and the functionalities of a relational database. Airtable aims to be suitable for a wide range of users, ranging from seasoned developers to non-technical individuals. One major attribute of Airtable is adaptability. Airtable lets its users set their own experience based on their proficiency with coding, thus allowing individuals with varying levels of coding expertise to develop applications. Airtable adopts a visual interface that simplifies the design and customization of the database, which reduces the need for extensive coding significantly. This largely simplified the development of the application using Airtable, Figure 4.3.

<table>
<thead>
<tr>
<th>Order</th>
<th>Description</th>
<th>Cost</th>
<th>Start date</th>
<th>Completed</th>
<th>Assigned</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>This is sample order 1</td>
<td>10.0</td>
<td>2023-10-27 16:00</td>
<td>☑️</td>
<td>Markus Weilin</td>
</tr>
<tr>
<td>2</td>
<td>This is sample order 2</td>
<td>20.0</td>
<td>2023-10-27 14:09</td>
<td>☑️</td>
<td>Friedrich Lehndorff</td>
</tr>
<tr>
<td>3</td>
<td>This is sample order 3</td>
<td>30.0</td>
<td>2023-10-27 14:10</td>
<td>☑️</td>
<td>Markus Weilin</td>
</tr>
<tr>
<td>4</td>
<td>This is sample order 4</td>
<td>40.0</td>
<td>2023-10-27 14:10</td>
<td>☑️</td>
<td>Friedrich Lehndorff</td>
</tr>
<tr>
<td>5</td>
<td>This is sample order 5</td>
<td>50.0</td>
<td>2023-10-27 14:10</td>
<td>☑️</td>
<td>Markus Weilin</td>
</tr>
<tr>
<td>6</td>
<td>This is sample order 6</td>
<td>60.0</td>
<td>2023-10-27 14:10</td>
<td>☑️</td>
<td>Friedrich Lehndorff</td>
</tr>
<tr>
<td>7</td>
<td>This is sample order 7</td>
<td>70.0</td>
<td>2023-10-27 14:10</td>
<td>☑️</td>
<td>Markus Weilin</td>
</tr>
<tr>
<td>8</td>
<td>This is sample order 8</td>
<td>80.0</td>
<td>2023-10-27 14:10</td>
<td>☑️</td>
<td>Friedrich Lehndorff</td>
</tr>
<tr>
<td>9</td>
<td>This is sample order 9</td>
<td>90.0</td>
<td>2023-10-27 14:10</td>
<td>☑️</td>
<td>Markus Weilin</td>
</tr>
<tr>
<td>10</td>
<td>This is sample order 10</td>
<td>100.0</td>
<td>2023-10-27 14:10</td>
<td>☑️</td>
<td>Friedrich Lehndorff</td>
</tr>
</tbody>
</table>

Figure 4.3: View of the application developed in Airtable platform
Chapter 5

Result and analysis

In the following chapter, the outcome of the evaluation process will be presented, with a particular focus on three selected LCDPs, Outsystems, Mendix, and Airtable. Each of the LCDPs was examined carefully with the established criteria in mind. The results were based on two ways to get a good comparison. SUS questionnaire was utilized to get a quantitative score of the platform’s qualities and features. The score given by the SUS questionnaire provided an insight into the user experience and usability of the platforms. Additionally, the performance between the LCDPs was measured and compared, which was a critical aspect of the total evaluation.

5.1 System Usability Scale

To promote a comparative analysis across the platforms, the SUS questionnaire was employed. This questionnaire was used to assign a quantitative score to the various features and qualities present on the development platforms. To get an even and fair assessment of the UX, the authors of this thesis evaluated the platforms independently, named "Evaluator 1" and "Evaluator 2" in the tables. The assigned scores (Table 5.1, 5.2, 5.3) were derived, and from these scores, an average score was subsequently computed. The results of this evaluation are presented in three distinct tables, featuring assessments of Mendix (Table 5.1), OutSystems (Table 5.2), and Airtable (Table 5.3), along with their respective scores. To achieve the overall score for each individual platform, the scores from the even and odd questions were computed in accordance with the established guidelines of SUS, which are explained in Section 3.3. As indicated in Table 5.3, the platform that received the highest score was Airtable. Determining what an acceptable usability score is can be
complex in some cases. The highest score was given to Airtable (Table 5.3). It can be a bit hard to define what result is accepted. However, according to Aaron Bangor, Philip T. Kortum, and James T. Miller, who conducted an empirical study of SUS, where data from nearly a decade was collected from various product evaluations that employed SUS as an assessment tool, a guideline was made. This research suggests that a SUS score above 70 is generally regarded as good usability [17].

<table>
<thead>
<tr>
<th>Mendix</th>
<th>Question number</th>
<th>Evaluator #1</th>
<th>Evaluator #2</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>I think that I would like to use this system frequently.</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2.</td>
<td>I found the system unnecessarily complex.</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3.</td>
<td>I thought the system was easy to use.</td>
<td>2</td>
<td>3</td>
<td>2.5</td>
</tr>
<tr>
<td>4.</td>
<td>Would I need the support of a technical person to be able to use this system.</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>5.</td>
<td>I found the various functions in this system were well integrated.</td>
<td>3</td>
<td>2</td>
<td>2.5</td>
</tr>
<tr>
<td>6.</td>
<td>I thought there was too much inconsistency in this system.</td>
<td>0</td>
<td>1</td>
<td>0.5</td>
</tr>
<tr>
<td>7.</td>
<td>I would imagine that most people would learn to use this system very quickly.</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>8.</td>
<td>I found the system very cumbersome to use.</td>
<td>1</td>
<td>2</td>
<td>1.5</td>
</tr>
<tr>
<td>9.</td>
<td>I felt very confident using the system.</td>
<td>3</td>
<td>2</td>
<td>2.5</td>
</tr>
<tr>
<td>10.</td>
<td>I needed to learn a lot of things before I could get going with this system.</td>
<td>2</td>
<td>3</td>
<td>2.5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Odd questions</th>
<th>Even questions</th>
<th>Total SUS-score</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>5.5</td>
<td>16.5</td>
<td>55</td>
</tr>
</tbody>
</table>

Table 5.1: SUS-score for Mendix platform
<table>
<thead>
<tr>
<th>Question number</th>
<th>Evaluator #1</th>
<th>Evaluator #2</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. I think that I would like to use this system frequently.</td>
<td>3</td>
<td>2</td>
<td>2.5</td>
</tr>
<tr>
<td>2. I found the system unnecessarily complex.</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3. I thought the system was easy to use.</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4. Would I need the support of a technical person to be able to use this system.</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>5. I found the various functions in this system were well integrated.</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>6. I thought there was too much inconsistency in this system.</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7. I would imagine that most people would learn to use this system very quickly.</td>
<td>1</td>
<td>2</td>
<td>1.5</td>
</tr>
<tr>
<td>8. I found the system very cumbersome to use.</td>
<td>2</td>
<td>1</td>
<td>1.5</td>
</tr>
<tr>
<td>9. I felt very confident using the system.</td>
<td>4</td>
<td>3</td>
<td>3.5</td>
</tr>
<tr>
<td>10. I needed to learn a lot of things before I could get going with this system.</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td><strong>Odd questions</strong></td>
<td><strong>7.5</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Even questions</strong></td>
<td></td>
<td><strong>16.5</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Total SUS-score</strong></td>
<td><strong>Odd questions</strong></td>
<td><strong>Even questions</strong></td>
<td><strong>60</strong></td>
</tr>
</tbody>
</table>

Table 5.2: SUS-score for OutSystems platform.
<table>
<thead>
<tr>
<th>Question number</th>
<th>Evaluator #1</th>
<th>Evaluator #2</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. I think that I would like to use this system frequently.</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2. I found the system unnecessarily complex.</td>
<td>1</td>
<td>0</td>
<td>0.5</td>
</tr>
<tr>
<td>3. I thought the system was easy to use.</td>
<td>3</td>
<td>4</td>
<td>3.5</td>
</tr>
<tr>
<td>4. Would I need the support of a technical person to be able to use this system.</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5. I found the various functions in this system were well integrated.</td>
<td>3</td>
<td>2</td>
<td>2.5</td>
</tr>
<tr>
<td>6. I thought there was too much inconsistency in this system.</td>
<td>0</td>
<td>1</td>
<td>0.5</td>
</tr>
<tr>
<td>7. I would imagine that most people would learn to use this system very quickly.</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>8. I found the system very cumbersome to use.</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>9. I felt very confident using the system.</td>
<td>3</td>
<td>4</td>
<td>3.5</td>
</tr>
<tr>
<td>10. I needed to learn a lot of things before I could get going with this system.</td>
<td>2</td>
<td>1</td>
<td>1.5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Odd questions</th>
<th>Even questions</th>
<th>Total SUS-score</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>9.5</td>
<td>20.5</td>
<td>75</td>
</tr>
</tbody>
</table>

Table 5.3: SUS-score for Airtable platform.
<table>
<thead>
<tr>
<th>Question</th>
<th>Mendix</th>
<th>OutSystems</th>
<th>Airtable</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. I think that I would like to use this system frequently</td>
<td>2</td>
<td>2.5</td>
<td>2</td>
</tr>
<tr>
<td>2. I found the system unnecessarily complex</td>
<td>1</td>
<td>1</td>
<td>0.5</td>
</tr>
<tr>
<td>3. I thought the system was easy to use</td>
<td>2.5</td>
<td>2</td>
<td>3.5</td>
</tr>
<tr>
<td>4. Would I need the support of a technical person to be able to use this system</td>
<td>3</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>5. I found the various functions in this system were well integrated</td>
<td>2.5</td>
<td>3</td>
<td>2.5</td>
</tr>
<tr>
<td>6. I thought there was too much inconsistency in this system</td>
<td>0.5</td>
<td>0</td>
<td>0.5</td>
</tr>
<tr>
<td>7. I would imagine that most people would learn to use this system very quickly</td>
<td>1</td>
<td>1.5</td>
<td>3</td>
</tr>
<tr>
<td>8. I found the system very cumbersome to use</td>
<td>1.5</td>
<td>1.5</td>
<td>1</td>
</tr>
<tr>
<td>9. I felt very confident using the system</td>
<td>2.5</td>
<td>3.5</td>
<td>3.5</td>
</tr>
<tr>
<td>10. I needed to learn a lot of things before I could get going with this system</td>
<td>2.5</td>
<td>3</td>
<td>1.5</td>
</tr>
</tbody>
</table>

| Total SUS-score             | 55      | 60         | 75       |

Table 5.4: SUS-score for all platforms.


5.1.1 Usability Evaluation

Looking at the SUS-score results in Table 5.4, we find that Airtable is the only platform with a score that would be deemed as "good." Mendix and OutSystems received an "ok" score, with OutSystems performing slightly better than Mendix. Based on these results, it would seem like Airtable is the clear winner. However, it must be taken into account that Airtable has a much more narrow focus in terms of its capabilities. Mendix and OutSystems offer superior flexibility in terms of what can be achieved on their respective platforms, and the flexibility comes at the expense of a steeper learning curve. As such, Airtable should not be portrayed as a direct competitor.

Between Mendix and OutSystems, the user has full control over the application being designed. Elements and functionality can be added freely and with ease. In both LCDPs, you can choose to start from a blank canvas or a template. Pre-built elements, or widgets, as both platforms call them, can then be placed in the application using drag-and-drop functionality. The widgets could, as an example, be text fields, lists, buttons, forms, and more. Widgets may then be connected to databases to display information or to handle some logic. Mendix calls its logic "Microflows" and "Nanoflows", where microflows are server-side, and nanoflows are client-side. OutSystems offer the same functionality but calls them "Client Actions" and "Server Actions." Common logic is often available, but more complex logic must be built manually. When constructing custom logic, it is visualized as a flow chart going from start to end and is built using pre-built components such as if-clauses, object modifications, and page redirections.

The platforms offer near-identical functionality, which corresponds to the results found by A. Sahay et al. [14]. The tangible difference in our experience was that OutSystems offered more control over how elements in the application interacted with each other. We also experienced that OutSystems offered superior guidance and documentation when learning the platform and troubleshooting, which is critical for new users.

5.2 Performance

After having run the tests, we end up with the following results for our web applications that were created using the selected platforms:
In our results for load time, we find that Mendix and OutSystems perform similarly in both Test 1 and Test 2 (Section 3.4), loading the contents of the page within a second of each other. However, in Test 3 (Section 3.4), OutSystems is able to complete the test significantly quicker than Mendix, only taking 1.99 seconds to complete on average, while Mendix took on average 11.04 seconds. This will be explored further below. As for Airtable, there is only data for Test 1, in which it took more than three times longer to load compared to both Mendix and OutSystems. We were unfortunately unable to gather data for Airtable in Test 2 due to our implementation requiring us to use a separate window to add orders, making it difficult to collect fair data, and in Test 3 since we did not manage to conveniently create a list of 1000 elements.

It is expected for Mendix and OutSystems to perform the same considering their similarities, and Airtables performance can be explained due to it being a web-based development platform, and additional content besides the created application being loaded.
Looking at Figure 5.2, the memory usage for the web applications can be seen. Again, we find that Mendix and OutSystems produce similar results in Test 1 and Test 2 (Section 3.4), and that there’s a large disparity in the results from Test 3 (Section 3.4). OutSystems outperforms Mendix by a large amount, using only one fifth of the memory compared to Mendix. Airtable is also using significantly more memory than the other two platforms in Test 1.

Since memory is being dynamically allocated as we navigate through the pages of the application, the maximum amount of memory usage at any given point was captured. Meaning that the average memory usage throughout the test could be widely different.

To better understand the results, a closer look at how the applications process and display the data is necessary. Looking at a separate performance trace that was captured as we were scrolling through the content of the application, we found that OutSystems is continuously processing scripts and rendering the information on the page, while Mendix has already processed all scripts and only has to render the information. This is shown in Figures 5.3 and 5.4, where the yellow area is for scripting, the purple is for rendering, green is for painting, and gray is for “other.” The same explanation can be given for memory usage. OutSystems dynamically allocates and releases memory as the content displayed on the page changes, resulting in significantly less memory usage.
usage.

Figure 5.3: Performance trace when scrolling through the Mendix application

Figure 5.4: Performance trace when scrolling through the OutSystems application

5.3 Discussion

What makes a LCDP better than another is difficult to define. In our results, Mendix and OutSystems performed similarly, except for a large discrepancy in Test 3 due to different strategies. While having a faster page load time is a desired property, the drawback is that the system may struggle to keep up with user actions, resulting in a less than ideal user experience. Airtable is also not a worse platform because of its limitations in comparison to other platforms; it achieves what it promises without any issues.

In order to further evaluate the platforms, ideally, the development of a more sophisticated application would be necessary to further examine all of the capabilities of the platforms. A more advanced application would impose a higher demand for features on the various LCDPs, thereby showing the eventual limitations they might have. Consequently, this would change the grading of the different platforms, both when comparing their respective SUS score and performance results. One potential problem that should be mentioned is that the SUS score is only based on two evaluators, which in this case are the authors of the thesis. Having a larger sample of testers with varying degrees of programming knowledge would increase the validity of the usability results.
Chapter 6

Conclusion

In this concluding chapter, a summary of the findings is presented along with the project’s limitations. Considerations for future work are also delineated, explaining what is left unaddressed in this project and what could be covered in continuous research.

6.1 Conclusion

Low-code has proven itself to be a good entrance gate when it comes to software development. Different platforms can be used for various purposes. However, there are still some questions that remain unanswered, especially when it comes to business-related questions. In Chapter 1, some questions regarding cost-effectiveness are asked and whether the usage of LCDPs can replace traditional development and coding. The foundation behind this question is that the low-code society is going by the term ”no to little coding experience needed.” This has shown itself to not be entirely true. As seen in Table 5.4, all three of the platforms greatly benefited from having previous knowledge when it came to the software development realm. It should be noted that Airtable was the easiest one to use. However, Airtable also had the most limitations. Airtable is the only platform that is considered to have good usability if only the scores are taken into account. However, there are more factors that should be considered when the final evaluation is done. Even though Mendix and OutSystem have a far deeper learning curve, they offer far more and various options for what could be done on their respective platforms. While Airtable is slightly limited to only letting users develop less advanced applications, OutSystems and Mendix are developed to handle more extensive and sophisticated tasks. In light of this, Mendix and OutSystems would likely
be awarded a higher SUS-score when working with a different application than the one developed for this thesis.

Regarding the performance of the low-code-created applications covered in Section 5.2, it was concluded that LCDPs with similar goals produce similarly performing applications. It was found that what sets the platforms performances apart is ultimately based on deliberate design decisions. OutSystems performed best in the tests, although this was achieved by only rendering what was visible to the user. This works great on a fast machine, though in other scenarios it may lead to stutters while navigating the page as new content is being processed. AirTable showed the worst performance of the three; however, AirTable is the only web-based development platform, and that alone has its strengths. In the end, performance should not necessarily be a deciding factor when it comes to choosing a LCDP. It is more important to value, for example, the functionality and usability of the platform.

The purpose of this thesis was also to find out if low-code was here to stay or merely a passing trend. With the experience gained, we conclude that it is here to stay. It may not replace traditional coding, as previously mentioned, and it will likely never exceed the potential of traditional coding. However, it is a great tool that can be used alongside traditional coding for many purposes. It is also a way to create competent solutions at a low cost and at a fast pace, making it a respectable choice for businesses of any size. It can also be a good entrance gate for people with no previous coding experience in software development. Allowing more people to raise their programming interest and knowledge.

### 6.2 Limitations

Due to the time limit of the project, it was necessary to have some criteria when choosing platforms. This made us only scratch the surface of the landscape of all the available platforms that are in use today. However, a good insight into how the platforms are performing was still given, which helped us achieve the goals that we aimed for.

One notable constraint that was identified was the cost. A significant number of platforms had a requirement of payment for access. The main criterion when selecting platforms was the prerequisite, which required the platforms to be free of charge. Consequently, numerous LCDPs were rejected as a result of that. However, this limitation prompted a more thorough exploration of available platforms, especially the identification of smaller platforms like Airtable. Airtable, which not only offered free development
capabilities but also proved to be highly suitable for the project when comparing different platforms.

6.3 Future Work

This project is only a first step and a scratch of the entire low-code landscape. The amount of LCDPs is growing rapidly, and our tests were only run on three different platforms. Recent reports also hint that vendors have started working on incorporating AI assistance into their platforms to enhance their products and aid developers [19]. In the future, more thorough research should be conducted where a larger number of platforms are included. This would lead to a broader result. Finally, many of the platforms have a free-of-charge trial where not all the features can be used. There is definitely an interest in evaluating a platform and its features to their full extent in order to further explore what is to be expected in the future of low-code.
References


