MASTER'S THESIS

Challenges Development Teams Face in Low-code Development Process

Alyousef, Z.

Award date:
2021

CHALLENGES DEVELOPMENT TEAMS FACE IN LOW-CODE DEVELOPMENT PROCESS

by

Zaher Alyousef

in partial fulfillment of the requirements for the degree of

Master of Science
in Software Engineering

at the Open University, Faculty of Science
Master Software Engineering

November 30, 2021

Student number:
Course code: IM9906
Thesis committee: **dr. ir. Fenia Aivaloglou** (primary supervisor), Open University
dr. Greg Alpár (secondary supervisor), Open University

CONTENTS

List of Tables

Acknowledgements

This is my last piece of writing in my thesis, and it brings all the memories back from struggling throughout the Covid-19 lock-down to working full-time from home with my family and kids playing around. I want to make sure to include all the people that helped me proceed this journey to the end.

First, I want to start by stating that I am grateful to my primary supervisor Fenia Aivaloglou who has provided me with outstanding guidance and valuable feedback every time I needed it, and to my secondary supervisor Greg Alpár for his constructive feedback.

Second, I would like to thank all my colleagues at KPMG for the support and help along the way, especially my development manager Dennis Stam.

Last but not least, I must express my heartfelt appreciation to my beloved family for providing me with unconditional support and continuous encouragement throughout my years of study, my wife Taimaa and my children Yaman, Amr, and the little one Taim.

Abstract

The low code programming paradigm has increased in popularity during the last few years. Several low code platform vendors advocate that it promotes development simplicity, accelerates the software development cycle and minimizes writing and maintaining code, enables the delivery of operational applications, and saves costs. Low-code development was not covered adequately in academic research. In this study, we explored what challenges the development team faces in the low-code development process and what impact these challenges could have on the maintainability and performance of applications. Then, we investigated how the development team addresses these challenges to reduce their impact. We followed an exploratory approach and used three data sources. We reviewed the related research and technical reports. Then, we explored the top discussions in low-code online forums. Finally, we conducted interviews with low-code developers. The results highlighted the challenges the developers faced in the low-code development process. The challenges were categorized into Background and Experience, Extensibility and Flexibility, Code-review/Version-control and Collaboration, Documentation and Support, Customization, Data Storage and Access, Integration, Testing, Technical debt and Refactoring, and Other Challenges. We found that the maintainability and performance of low-code applications could be impacted when the development team is mainly citizen developers, the application is complex, implements a non-supported use case, there is no code review, no/amber version control management, or not following efficient development framework methodologies. Additionally, practices such as extending applications with traditional code, customizing an existing workflow or component to implement complex logic, creating an inconsistent data model/architecture, and writing complex data queries could decline maintainability and performance. Furthermore, the development teams we interviewed took some actions to address the challenges, such as educating the citizen developers about the development principles, setting some strategies to handle the versioning and code-review, and following best practices. Some LCDPs have inbuilt mechanisms that analyze the development output quality, which could help in improving the software quality.

1

INTRODUCTION

Traditional software development methods often fail to deliver within a fixed time frame and meet the fast-changing requirements. Moreover, keeping up with the essential maintenance and the required extensibility in legacy systems can be challenging. It has been found that approximately 70% of development projects between 2011 and 2015 were considered unsuccessful by stakeholders Hujainah et al. [2018]. During 2017, was also found that the estimated percentage of failed projects escalated, where only 29% of projects met the financial plan, and 20% were completed on time Öbrand et al. [2019].

At the same time, the digital transformation wave led to a significant rise in the demand for software and software developers. Additionally, supplying business automation services is impacted by developers' shortages and lacking the required skills. Software delivery failure could prevent organizations from the advantages of the competitive aspect. Hence, seeking alternatives and techniques that can solve software delivery barriers became vital.

1.1. ALTERNATIVE SOFTWARE DEVELOPMENT METHODOLOGIES

Methodologies and practices such as Rapid Application Development (RAD), Agile, and DevOps emerged to eliminate the constraints and flaws of the traditional development and management methods like the waterfall model. Agile methodologies such as Scrum, Kanban, Crystal, Extreme Programming, and Feature Driven Development focus on agility and adaptability. Agile approach utilizes multiple small iterative development cycles, called sprints, where each sprint is a complete development cycle. Customers engage by providing their feedback on a demonstration given at the end of every sprint. This feedback may be an input for the next course of changes in the next sprint, till a product that exactly meets the customers' expectations is delivered McCormick [2012]. RAD focuses on faster development within budget without sacrificing system quality Fitzgerald [1999]. It is a rapid prototyping methodology for developers to change their development process and improve development capabilities. The development process in RAD is an iterative sequence of short development cycles to boost productivity, efficiency, and self-correcting. It reinforces the collaboration between developers, end-users, and other stakeholders Berger et al. [2004]. RAD could be considered as a form of Agile methodology that focuses on prioritizing prototype iterations or releases. DevOps brings developers and operations together to deliver reliable software and services quickly, with quality built-in. DevOps practices are

broad and focus on different aspects such as culture, collaboration, automation, measurement, and monitoring Mishra and Otaiwi [2020]. These development methodologies and practices could be used in their standard model or combined in a hybrid model that best suits the organization. However, while they speed up the development process, writing and maintaining code is still the most time-consuming activity. Software developers spend 39% of their time writing new code or improving existing code. They also spend 22% of their time just doing code maintenance Grams [2019].

1.2. Low-code Development

Visual development could speed up software development and skip or minimize writing and maintaining code as an alternative to code writing. In 2011 [1], some RAD development tools that facilitate the visual development and model-driven design appeared in the market. Later in 2014, Clay Richardson and John Rymer invented the term "Low-code" to classify these platforms in Forrester's report "New Development Platforms Emerge For Customer-Facing Applications" Richardson et al. [2014].

Gartner[2] defines a low-code application platform (LCAP) as "an application platform that supports rapid application development, one-step deployment, execution, and management using declarative, high-level programming abstractions, such as model-driven and metadata-based programming languages." A Low-code development platform (LCDP) is software provided on the cloud through a Platform-as-a-Service (PaaS) model and serves in rapidly building and delivering applications that take advantage of the cloud infrastructures. LCDP turns the application development process from manual coding into interacting with graphical user interfaces, using premade components, configurable settings, model-driven logic, and different actions such as drag-and-drop. They are prepared for users with no appropriate background in programming or software development, called citizen developers in the LCDP language Tisi et al. [2019]. According to Gartner [3] "A citizen developer is an employee who creates application capabilities for consumption by themselves or others, using tools that are not actively forbidden by IT or business units. All citizen developers are business technologists". Citizen developers attend special training afforded by the LCDP vendor to be capable to use that LCDP. Low-code reinforces and improves the Agile mindset and methodologies since they both converge in all Agile main principles.[4] Furthermore, some low-code platforms provide DevOps tools out of the box, such as continuous integration and continuous delivery (CI/CD), test automation, and monitoring.[5]

The low-code development process could deviate from the traditional one in many aspects, such as defining requirements, testing and automation of DevOps, support, and maintenance Sulak [2020]. The development teams could be composed differently based on the skill-set aspect, and the roles might differ since there is a new role –citizen-developer–

[1] sdtimes.com/application-development/low-code-development-seeks-accelerate-software-delivery/
[2] www.gartner.com/en/documents/3970417
[3] https://www.gartner.com/en/information-technology/glossary/citizen-developer
[4] www.mendix.com/blog/low-code-principle-3-agility
[5] www.evaluation-guide.mendix.com/evaluation-guide/app-lifecycle/devops

that has been introduced.[6] Low-code testing (functional and non-functional) could have altered characters regarding three concerns: the citizen developer function in testing, performing high-level test automation, and cloud testing Khorram et al. [2020]. The same applies to the design process, prototyping, estimation approach, versioning and branching,[7] and the automation of some methods such as CI/CD pipelines. Software development is not only fast application delivery, but it is also making sure the applications present reusability and maintainability so that total life-cycle costs are reduced Agarwal et al. [2000]. The velocity in low-code development and being used by citizen developers could cause code-smells and bad patterns, for example, which influence the application's maintainability Dijkink [2020].

There is not much academic research about using low-code platforms. The state of the art in low-code development comes from independent research companies that provide regular technical reports about this field. We explored some of the recently published reports by Gartner and Forrester and used them as references. We focused on Forrester [8] and Gartner [9] as they are the most famous in this field.

Low code development can be an effective solution to software development disorder due to the lack of developers and required skills. According to a 2020 Gartner report, "The vendors of low-code application platforms have been improving the ease at which business applications can be delivered, providing broader capabilities requiring smaller and less specialized teams of developers" Vincent et al. [2020]. LCDPs are increasingly replacing the development of standard business applications in Java or .NET and offer distinctive alternatives to commercial off-the-shelf or SaaS applications. Gartner highlighted five use cases addressed by LCDPs in its report Vincent et al. [2020]. They are: support a citizen development strategy, delivering web and mobile business unit applications, building enterprise IT business process applications, developing composite applications by fusion (multidisciplinary) teams, and building web and mobile SaaS and ISV (independent software vendor) applications.

Low-code platforms proved their importance and ability to play a vital role in software development. Gartner predicts that "by 2024, low-code application development will be responsible for more than 65% of application development activity". Gartner also classified some low-code vendors such as Mendix, OutSystems, Salesforce, Microsoft, ServiceNow, and Appian as leaders in this field.

1.3. Low-code Benefits and Drawbacks

Low-code promotes development simplicity and ease of use, enables and accelerates the delivery of fully or partially operational applications, diminishes the hand-coding, helps individuals other than programmers to contribute to the development of applications, and saves costs in setup, training, deployment, and maintenance Richardson et al. [2014]. Low-code developers –traditional developers and citizen developers– can be involved in the development and maintenance activities and apply the required modifications directly. Low-

[6] marutitech.com/no-code-low-code-vs-traditional-development/
[7] www.outsystems.com/evaluation-guide/how-does-outsystems-enable-team-collaboration
[8] www.forrester.com/research/
[9] www.gartner.com/en

code development involves fewer engineering efforts, such as manual coding, deployment, and infrastructure management, which lowers down the costs of the development process compared to the traditional development Koksal [2020].

On the other hand, low-code is not drawbacks-free. Some experts such as Leon van Heerden [10], Peter Wayner [11], and Matt Heuser [12] wrote articles or blogs where they highlighted some challenges in low-code. For example, the modular components approach restricts applications' customization. Additionally, integration with other systems (especially with legacy systems) is limited and causes technical problems. Some low-code development platforms tie you up with their cloud-based offering and make it challenging to customize your application. Peter Wayner talked about developers' frustration with low-code, which could grow because of the maintenance difficulty, no surprise or novelty, getting stuck with "one size fits all", the configuration could be more complicated than coding, inefficiency in some cases, and the lack of experience.

1.4. Proposed Research

Software quality is important, especially the maintainability and performance, as software systems are constantly subject to change and extension. Similarly, the visually developed applications with any LCDP are subject to change. They might go through customization, integration, or extension. The software quality aspects such as maintainability and performance should be taken into account while working with low-code. Complex or large applications are tough to maintain, especially if they contain long, complex, or duplicated workflows that are hard to analyze. A weak architecture that creates a lot of dependencies between workflows and components is another example [13]. Low-code is majorly used by citizen developers who have no IT background. This could result in applications with low maintainability and performance. There will be no software engineering conventions followed when designing the architectures and doing the implementation.

Some coding exercises are different in low-code development, for example, writing code, code review, maintenance, and refactoring. Considering the differences, starting low-code development could be challenging, especially if there is no low-code expertise in the team. Moreover, using LCDPs to build large applications and other vital applications makes software quality a priority. Some potential challenges could hinder the low-code development process, affect the software quality. Therefore, we need to investigate these challenges and find out how to deal with them to prevent any detrimental effects.

Despite its increased popularity nowadays, low-code development has not been well covered in academic research. You can only find a few studies about related topics. For example, studies on specific applications of low-code in the manufacturing domain Sanchis et al. [2019], and the healthcare sector Ness and Hansen [2020], study on comparison of LCDPs Sattar [2018], and another on about developers' experience in low-code development environments Dahlberg [2020]. In Chapter 2, we present an overview of the related

[10] https://journeyapps.com/blog/10-challenges-facing-low-code-platforms/
[11] https://www.infoworld.com/article/3438819/why-developers-hate-low-code.html
[12] https://searchsoftwarequality.techtarget.com/feature/A-low-code-platform-can-do-a-lot-but-it-has-limits
[13] https://www.softwareimprovementgroup.com/resources/low-code-shouldnt-mean-low-quality/

work we could find.

The proposed research concerns investigating the potential challenges development teams face in the low-code development process, how these challenges could influence the software quality from maintainability and performance aspects, and how the development teams remediate them. We try to focus on the low-code development process, looking into aspects such as the coding routines (implementation, code-review, testing, customization). The platform facilitation of the development process and collaboration (support, documentation, version-control management, coordination). The citizen developer role and her contribution to the development process, taking into account that she has no knowledge about software engineering principles and how that could affect her work quality in customizing and extending software solutions.

We follow three steps in this study to collect and analyze the required data for our research. Firstly, by reviewing the related literature. Next, we explore some low-code forums. Lastly, the essential part of the research is conducting interviews with low-code developers from KPMG and some other organizations.

2

Research Context

The low-code software development approach aims to minimize manual coding compared to traditional development and reduce the workloads by dividing the IT backlog between the citizen developers and the professional software developers. Although low-code platforms appeared in 2011 [1], we only found a few studies that explore the low-code development process and provide tangible results from the field.

2.1. Background on Low-code Development

Here we explore some literature about low-code-related topics.

2.1.1. Enabler of Digital Transformation

Software systems should be flexible when reflecting on the new market requirements to adapt to the current digital transformation wave. The priorities are to produce innovative new products and services, ensure the IT performance is consistent and stable, enhance business processes, improve the customer experience and work efficiency. These priorities result in growing backlogs and demand for developers. Low-code software development has emerged as a solution to these obstacles Tiemens et al. [2019]. It has been applied in various industries. Sanchis et al. [2019] focuses on the current characterization of the low-code domain, following the foundations of the computer-aided software engineering field. The paper uses a theory-building research methodology to analyze research related to the low-code development platforms. It discusses and analyzes the benchmarking of the current LCDPs in the manufacturing industry to identify the missing features.

Waszkowski [2019] describes the use of a low-code platform Aurea BPM for automating business processes in the manufacturing domain to resolve problems such as the shortage of programmers and growing requirements. The paper concludes that low-code platforms can significantly reduce the cost and time of implementing, developing, and maintaining processes. Moreover, it was found that employing the company's internal human resources in the analytical and development activities made the lack of programmers and increasing requirements obstacles disappear.

[1] sdtimes.com/application-development/low-code-development-seeks-accelerate-software-delivery/

In the healthcare sector, Ness and Hansen [2020] studied the use of low-code to develop an electronic health record (E)HR for central healthcare services and an interactive platform for all healthcare services in Norway. The findings revealed that the high speed, adaptability, and innovation in low-code development contributed to solving challenges such as the time restriction and flexibility of the proposed solution. Adaptability helps to hold the software up to date and reduces the chance of ending in obsolete technology. The study concluded that low code could make it possible to re-customize the software for each healthcare service and municipality's needs without affecting the core functionalities.

2.1.2. Comparison of LCDPs

There are currently many low-code development platforms on the market. Sattar [2018] explores LCDPs to provide a decision tree to decide which low-code platform to select for a particular problem based on a given organization and the application type. Sahay et al. [2020] also provides a comprehensive classification of the current LCDPs to enhance the users' familiarity in this area and facilitate the users' adoption of platforms based on their requirements. The paper presents a technical review and analyzes eight representative LCDPs. Both papers assist in selecting the suitable low-code platform for a specific use case to avoid any potential challenges.

2.2. Low-code Development Process and Software Quality

The number of studies on the low-code development process and code quality is limited. Here we will explore some studies on aspects associated with our main topic.

2.2.1. Challenges in Low-code Development

Compared to traditional development, low-code platforms change the development aspects (scope, development, maintenance, integration, time to market, and deployment), enabling developers to write less code. As a result, the low-code development process provides a different experience for the developers than their experience in other traditional development processes. Knowing the developers' experience in low-code could highlight some of the challenges they deal with. Dahlberg [2020] investigated the developers' experience when working with low-code. The study introduces a case study from an IT company that started using low-code recently. In the case study, the developers have experience in both low-code and traditional development. From the study findings, the main positive experiences about low-code development are: feeling more productive, focusing on the task and development objective, quick learnability and onboarding, easy showing progress and prototypes, less effort spent on some required tasks after being automated. On the other hand, the reported negative experiences were: less freedom and creativity because of the limitations and restrictions set by the platform, inadequate documentation, and ineffectual teamwork due to no optimal collaboration features.

The master's thesis of Virta [2018] looks into how low-code development is different from traditional software development and what the benefits and pitfalls are. The study reports on the employees of a Finnish Salesforce consulting company called Biit Oy, which started using low-code as an option to develop software. Salesforce is a cloud-based CRM

(customer relationship management) platform that provides free-of-charge low-code development tools to users. The study interviews eleven IT employees (consultants, developers, and architects) to collect qualitative data related to low-code benefits and pitfalls to create a comprehensive list and validate what has been proposed by vendors. Some of the low-code challenges from the study findings are the inefficiency of the low-code solutions, major performance issues with large data volumes, the lack of ability to bulkify database calls, maintenance issues with more complex solutions, and the obscurity of the offered tools. The study reports some pitfalls of low-code development in Salesforce. It Lacks some common data structures such as maps (also known as dictionary or associative array). It is not easy to create complex solutions (using multiple classes or inheritance). The low-code developers may build weakly designed and impossible to handle applications as they are unfamiliar with software engineering principles. Moreover, there is a lack of skilled developers in this field, there is no support of version control, and no possibility to create comments inside the solution.

In another study (also master's thesis) Vikebø [2020], a low-code solution has been developed to automate the process of deferment of study-offer at the Admissions Office of the Norwegian School of Economics. The researchers used the Business Process Model and Notation Framework to document the process, and they developed an artifact using the low-code development platform OutSystems. Industry experts have evaluated the artifact to provide insights into the viability of low-code development platforms in the administration of institutions of higher education. One challenge found by this study was that most low-code development processes are still dependent on IT professionals running the underlying IT infrastructure.

2.2.2. Low-code Applications Quality

Low-code proposes new concepts and styles in the development process and quality assurance aspects such as testing and code review. Nevertheless, academic research does not sufficiently cover the concept of testing low-code software. Khorram et al. [2020] studies the existing challenges and opportunities when testing low-code and presents a specific definition Low-Code Testing. The paper focuses on five commercial LCDPs and analyzes their testing components to suggest low-code testing improvements from a business perspective. As a result of the analysis, the study proposes a feature list for low-code testing with the potential benefits (taking the low-code principles into account). The feature list can help in comparing or building low-code testing components. The paper presents the challenges of low-code testing regarding three areas: the citizen developers' role in testing, the importance of high-level test automation, and cloud testing. The complete engagement of citizen developers in the low-code testing activities is fundamental in all phases. Yet, many challenges arise because of their low technical knowledge, which requires new support techniques. Test automation is valuable and should be performed on a high level, alongside a soft dependency on technical expertise. However, most of the automated testing approaches were found to be very technical, and they depend on manual scripting.

Attention to code quality and focus on maintainability is very important in software development. One way to measure the maintainability of an application is to look for code smells. Code-smell indicates a deep-rooted problem in the code. These pointers are no

specific guidelines, but some structures propose the feasibility of refactoring Fowler [2018]. In her master's thesis, Dijkink [2020] investigates the occurrence of code smells in applications built with low-code platforms and how developers recognize them. The research approach was to collect and analyze qualitative data on code smells from applications built with OutSystems, as well as to conduct a qualitative survey to see what developers perceive as frequent code smells and harmful patterns and what risks they could cause. A list of code smells that occur in low-code has been completed from the final results. The most common harmful patterns are lack of module classification, cyclical references between applications, and missing description by public element. The most common code smells are data modules, large action, and duplicated code. The developers have seen some new code smells in low-code and considered that the most harmful, from their point of view, are duplicated code, large modules, and incomplete library modules.

As we have seen so far, few studies experiment using low-code in specific domains to see whether it can solve the obstacles in these domains and what is the best LCDP to use in a given area and use case. Few other studies investigated developers' experience in low-code, pitfalls they face, testing, and code-smells in low-code. It is vital to obtain a solid knowledge of the low-code development process, how developers work with it, and the challenges they face in this field. Since software quality is necessary to build a successful application, it is important to investigate what impact these challenges have on quality and how to resolve that. Low-code development enables citizen developers to develop software, which is still software engineering; this involves the same best practices of traditional development. Failure to apply them can lead to technical debt and damage the maintainability and performance of the built applications.

3

Research Questions

Some potential challenges could hinder the low-code development process and affect the maintainability and performance of applications, especially when doing customization, integration, or extension. Complex or big applications are hard to maintain, particularly if they include long, complicated, or duplicated workflows that are difficult to analyze. The citizen developers could introduce work that does not adhere to the software development principles, which causes low maintainability and performance. Some development practices could not be supported well in LCDPs, such as code-review and testing. Maintainability and performance should be regarded while working with low-code to build enterprise applications. Moreover, the low-code development process has not been covered adequately in academic research. Therefore, it is necessary to investigate these challenges and discover how to deal with them to maintain good quality.

3.1. Research Questions

In this study, we investigate the challenges that Low-code development teams could experience during the development process. We try to discover how these challenges could hinder the Low-code development process and affect the software quality from maintainability and performance perspectives and how low-code developers try to remediate that. To focus on the research goals, we form some questions that represent them. The research questions we aim to answer are as follows:

RQ1 What challenges could a development team face when working in the low-code development process?

RQ2 What impact could these challenges have on the software maintainability and performance?

RQ3 How does the development team deal with these challenges to reduce their impact?

3.2. Scope of Study

In this study, we focus more on two low-code platforms, OutSystems and Mendix, as leading platforms in the market, and there will be different platforms added as secondary targets. The reasons are that OutSystems and Mendix are mature platforms (Gartner named

them as leaders in the low-code quadrant Vincent et al. [2020]), these two platforms are used by KPMG where we do our study, and because of the limited time of the research.

4

Research Method

To answer our research question, we used three sources of data. As a start, we explored some low-code-related studies and looked for any reported challenges. Then, we explored two online low-code communities to help categorize our findings and define the focus areas for the interviews. As the last step, we conducted interviews with low-code developers. We used the output from the first two steps to formulate the interview questions to achieve the desired results. We asked the interviewees about the challenges they faced while working with low-code and checked whether they confirmed what we found from the previous steps.

4.1. Data Collection and Analysis

4.1.1. Exploring Related Work and Technical Reports

We analyzed academic studies on applying low-code in some sectors and on low-code quality-related aspects. Besides that, we explored technical reports on low-code published by Gartner and Forrester as a source for the state-of-art in the low-code field.
The work steps were:

1 We looked on Google Scholar and the OU library for the publications that contain keywords related to our research. For example, low-code development, low-code quality, low-code platforms, low-code challenges, and rapid application development. We also scanned the references of the publications we found.

2 We identified the most related publications by reading the abstract and conclusion sections, then we explored them and extracted all paragraphs that mention anything related to our research questions.

3 Created an Excel template to classify and label the extracted paragraphs. First, we analyzed each paragraph and gave it a preliminary label. Then, we grouped the labels that are close to each other under one generic label. The generic labels we got were: Experience, Extensibility, Collaboration, Documentation, Customization, Data storage and access, Integration, Code-review, Version Control, Testing, and Technical debt.

4 Summarized all the paragraphs that go under one label and cited them for using them later in our research results.

We presented the results from this step in Chapter 5 Research Results.

4.1.2. Exploring Low-code Online Communities

Most low-code vendors have online communities; official web forums the low-code practitioners use to seek/offer help by joining ongoing discussions. We scanned two of them, The OutSystems Community,[1] and the Mendix Community.[2] The essential purpose of exploring the low-code communities is to gain insight into potential topics of the low-code practitioners' discussion or questions that are related to our research questions. Therefore, we limited the analysis to the high-level topics or tags and did not go deep after the questions in these communities. The low-code communities we explored were using post-tagging mechanisms, so we looked into the most popular tags or topics to achieve our purpose. We collected data from two low-code online communities and then analyzed the gathered data to classify the challenges that low-code practitioners face.

To process the data we collected from the low-code communities, we followed five steps:

1 From each low-code community, we selected the most popular tags.

2 For each tag, we collected and analyzed data from the top five popular questions/discussions and used it to add a description to each tag to explain it and make it clear what it means. We looked also into the vendor's documentation to enhance the description.

3 Depending on the description, we refined the tags, merged what is similar, and kept what is related to our research focus on the development process. (See Table 4.1 and Table 4.2)

4 We bound each tag with a generic label depending on the related description to make it more abstract and comparable to the tags from the other low-code community. For example, Java in Mendix and C# in OutSystems are used to extend solutions so any tag about these languages could be bound with the generic label "Extensibility". In some cases, we connect a tag with more than one generic label, so we use / to indicate that.

5 We compared the tags from both tables (Table 4.1 and Table 4.2) and created a new table (Table 5.1) with the given generic labels that belong to both tables.

OutSystems Community

Regarding the OutSystems community, we collected the data we need from forums/tags[3]. On 3 July 2021, there were 121 records of tags with a different number of posts from 2 to 1976 per tag. We looked at the top 15 and extracted what is related to our research topic. Table 4.1 lists the top used ones.

[1] www.outsystems.com/community
[2] www.community.mendix.com/p/community
[3] www.outsystems.com/forums/tags

MENDIX COMMUNITY

From the Mendix community, we collected the data we need from the questions tab[4]. On 3 July 2021, there were 32799 questions. We looked into the top 15 used tags (sorted by usage) and extracted what is related to our research topic. The most popular tags are shown in Table 4.2.

[4] www.forum.mendix.com/p/questions

Tag	Generic label	Description
Forge	Extensibility / Customization	A repository of reusable, open code modules, connectors, and UI components. Discussions were about properties/methods support, and the introduction of reusable modules/components.
OutSystems Widgets	Extensibility / Customization	Reusable patterns, functionalities, or an interface component enable users to perform a function or access a service. Discussions about doing adjustments to the components.
Reactive / Traditional web	Extensibility / Customization	For questions about Reactive and Traditional Web App Development. The techniques that can be used to build sites that feel fast and responsive to user input. Discussions about responsive/ auto-complete/ scripting/ jQuery/ upload and process files/ filtering data/ Azure DevOps pipelines/ functions.
Development	Experience / Data Storage and Access	Creating, building, and maintaining applications or reusable components. Discussions about issues related to SQL queries, Errors, advanced statements/widgets, scripting.
Database / Aggregate	Data storage and access	Discussions about data modeling, aggregation, CRUD, fetch data using optimized queries.
API	Extensibility / Integration	With APIs, you can integrate your applications with external systems. Questions about errors while consuming REST API, connecting to external systems, request and response.
JavaScript	Extensibility / Customization	you can use JavaScript in OutSystems. It is part of the Extensibility and Integration of OutSystems. Questions are mainly about customization and styling.
New to OutSystems / Beginner	Experience	Questions and discussions around topics for new members to OutSystems or just starting to learn OutSystems, such as showing own work, applying validation, doing aggregation, training materials, using widgets, dealing with SQL queries ad database.

Table 4.1: The most used tags in the OutSystems community

Tag	Generic label	Description
Microflow	Experience	A visual way of expressing a textual program code. Perform actions such as creating and changing objects, showing pages, making choices, changing/extending the standard behavior, adding custom logic, and integrating with other systems. Questions were about how to implement with Microflow.
Widget	Extensibility / Customization	A part of the user interface in a Mendix app enable functionality and Interaction with the app users. Questions were about integrating these components into the app and fixing related issues.
Java / Java-action	Extensibility	Extend an application with custom Java code. You can extend the functionality of an application with Java actions, where it is not possible in microflows. Questions were about how to use, debug and fix Java custom functionality.
Webservice	Extensibility and Integration	Web services are preferred for integrating external systems with the Mendix application. Questions were about configuration, import, export, and input into a web service.
Database / Data storage and access	Data storage and access	Data can be represented by domain models. Every module in an app project can have its domain model, consisting of one or more entities. Mendix automatically creates tables to store your entities in the database. Using controls of the data grid you can show a list of objects in a table format, browse, search, and edit those objects. Questions were about how to connect, query, interact, and SQL.
XPath	Data storage and access	Mendix query language made to retrieve data. It uses path expressions to select data of Mendix objects and their attributes or associations. Questions about how to render custom queries and related issues.
JavaScript and CSS	Customization	The JavaScript Snippet widget help add a piece of JavaScript to a page or enhance styling by adding a CSS file into your project theme/CSS folder. Questions were about functionality or style customization, performance issues, the behavior of JS actions.

Table 4.2: The most used tags in the Mendix community

4.2. Conducting Interviews

As the last and most important step, we conducted interviews in order to collect qualitative data and answer our research questions. The results we gathered from the two data sources previously analyzed were used as input for the interviews and helped in forming the interview protocol. We asked the interviewees about the challenges they faced while working with low-code, and we checked if they confirm what we found from the previous steps. We discussed their point of view on how these challenges influence the software quality (maintainability and performance) and what they had done to fix that.

4.2.1. Interview Approach

We used a semi-structured interview protocol to allow for follow-up questions and get comprehensive answers as they are more elastic and go deeper than constructed interviews. In our interview protocol Jacob and Furgerson [2012], the questions were derived from the research focus area and the data we collected from the previous steps, which helped to narrow our questions and extract meaningful data.

We developed a script (see Chapter Appendix 7) to guide the interview workflow, starting with the basics, such as warming up the participants with required information about the research, explaining the consent form, and alleviating any concerns about confidentiality and privacy. The interviews were between 30 and 45 minutes; they started with easy-to-answer questions and then moved towards more difficult or argumentative open-ended ones. We had to apply for a cETO[5] approval to start with the interviews. Which is an assessment of the ethical aspects of a study, usually for OU researchers, focuses on participant information, informed consent, personal data protection act. It took a few weeks to complete and get the approval. The interviews were performed online considering the Covid-19 situation and the flexibility of online meetings. In compliance with the OU policy on audio and video recordings, we used Skype for the online recordings. We stored the audio/video data in a password-secured folder in a Research drive. We deleted the original data immediately after uploading them to the Research drive (a secure OU folder). Then, we used the files on the Research drive to transcribe the audio/video data so that persons are no longer traceable to a specific person. To comply with legislation and guidelines during research, we stored the study data and the collected data on a secure Research drive of OU. The consent form was sent as a PDF file via email to each participant before the interview, and we could collect either electronically signed or scan hand-signed PDF files. To get an organized interview, we set a structure for the questions. The first part was for the intro and opening with the easy-to-answer questions. The second part (more difficult or argumentative open-ended questions) had a list of topics (generic labels and tags we extracted in steps 1 and 2). For each topic, we discussed what challenges the participant faced, their influence on maintainability and performance, and how they mitigated that. Then, we checked if they faced any other challenges. The last part was to collect recommendations for the low-code practitioners/vendors and then close the interview.

[5] https://ceto.ou.nl/

	Main role	IT Education	Experience in low-code	Low-code platforms
P1	Traditional developer	Yes	3+ years	OutSystems, Mendix
P2	Citizen developer	No	3+ years	OutSystems, Mendix, Blueriq and Betty Blocks
P3	Citizen developer	No	10+ years	Mendix
P4	Traditional developer	Yes	10+ years	OutSystems
P5	Citizen developer	No	2+ years	Alteryx, Dataiku, Tableau prep
P6	Citizen developer with basic knowledge in programming	Subject close to IT	3+ years	Appian

Table 4.3: Participants information

4.2.2. Selection of Participants

The study took place mainly at KPMG, a big consulting firm in the Netherlands. KPMG is a global group of independent member firms offering audit, tax, and advisory services. Within Advisory and as part of the Digital Transformation suite there is the Digital Enablement team where they use both the traditional development technologies and low-code platforms (OutSystems and Mendix) to build applications tailored to clients' requirements.

We conducted six semi-structured interviews with six participants who have been working with low code development platforms for more than three years. The participants played different roles such as architects, testers, citizen developers, developers, lead/managers. To ensure that the participants had sufficient experience to report on, we looked for people who worked with different LCDPs such as OutSystems, Mendix, and other LCDPs, and had adequate experience with Low-code development. Two traditional developers, one with basic coding skills, and three citizen developers. Furthermore, the participants were from KPMG and other companies that work with low-code development platforms. We reached out to the participants via our professional network. We asked the first level connection to forward the invitation to other potential participants and contact the researcher for more information. This way, we did not collect personal information of participants not interested in participating. In Table 4.4, we include the profile information of each participant in this research. We refer to a participant with "P" and add a digit to recognize which one, for example, the third participant is "P3".

4.2.3. Analysis of Interview Data

We conducted a thematic analysis on the transcripts with three themes Braun and Clarke [2006], one for each research question. These themes were directly inferred from the research concern and guided by appropriate labeling from the findings of the other two information sources Blair [2015], Braun and Clarke [2006]. First, we determined the themes and then developed the labels. The themes were also in line with the main topics of the interview script (Challenges in Low-code Development Process, Impact on Maintainability and Performance, and Actions to Mitigate Challenges). Only one label (Other Challenges) was derived from the interviews data, and the rest of the labels were developed from the

findings of exploring the literature and low-code online communities (Background and Experience, Extensibility and Flexibility, Code-review/Version-control and Collaboration, Documentation and Support, Customization, Data Storage and Access, Integration, Testing, and Technical debt and refactoring). Second, we used these labels to code all interviews' transcripts and transferred them into an excel sheet to make it easier to process and filter the data. We created one sheet for each participant, where we added three columns representing the themes, and each label represented in one row. We filled these excel sheets with the summarized data from the transcripts. Third, we combined the data across participants for each theme per label, refined the final information, and used it in Section 5 Research Results.

5

Research Results

This chapter presents the results we derived from the data collection and analysis. Here we display the results per research question. Exploring the online communities focused on finding generic labels to classify the challenges in low-code development. The findings from these online communities are presented in Table 5.1 only and we will not mention them when presenting the findings from the other data sources. The generic labels we created from exploring the related work cover more development aspects such as code-review, collaboration, and testing.

Generic label	Description
Extensibility and Integration	Integrating external technologies, support of extensions, related properties, and functions, mainly questions about errors in request/response or consuming REST API, how to use, debug, and fix custom external code.
Experience	Challenges about advanced exercises related to software development principles, and doing the early real work during the development cycle, questions about best practices in error handling, questions about applying advanced functionality, and dealing with input.
Data storage and access	Questions about Data modeling, (optimized) queries, CRUD, connect and interact, and using SQL or other techniques to query data.
Customization	Challenges with scripting and styling, performance issues related to customization, and behavior of JS actions. Using formula fields, validation, and customization.

Table 5.1: The common challenges from both OutSystems and Mendix communities

We combined the generic labels from the related work with the generic labels from the online communities to have a final set of generic labels to use in structuring the interviews and presenting the research findings. The labels that are close to each other, such as Code-review/Version-control and Collaboration merged into one generic label. We added one extra generic label "Other Challenges" to represent the challenges that were added by the participants and do not belong to the other generic labels. In the end, ten generic labels were created as a result of the three data sources. The final generic labels are Background and Experience, Extensibility and Flexibility, Code-review/Version-control and Collaboration, Documentation and Support, Customization, Data Storage and Access, Integration, Testing, Technical debt and refactoring, and Other Challenges. This helps to sort out the reported challenges for the ease of following and processing.

5.1. RQ1: What challenges could the development team face when working in the low-code development process?

5.1.1. Background and Experience

One of the Low-code main goals is to enable the citizen-developers to build applications without prerequisite development experience. However, this was reported by some papers and technical reports as a challenge that prevents achieving successful software development. There must be someone in the team who is experienced with the principles of software engineering. Otherwise, the resulting systems could be poorly structured and designed that are difficult to manage, support, or maintain Virta [2018]. Even though low-code is for everyone, skilled developers are irreplaceable as they develop the most advanced systems and their core. LCDPs can enable non-professional developers to contribute to application development, but most low-code development processes still rely on IT professionals. Low-code characteristics can only reduce the need for skilled developers Vikebø [2020].

According to Gartner, Appian's low-code development platform is more suitable for professional developers. Appian's proprietary expression and the scripting language are typically inhibitors for citizen developers when building algorithmic expressions Vincent et al. [2020]. Some LCDPs use advanced programming languages such as SQL, making it difficult for low-code developers to work with it and deliver high performance. The learning process, on the other hand, does not cover these advanced technologies Dahlberg [2020].

The results from the interviews were confirming what we found in the literature. Two participants (P1 and P4) –the traditional developers– did not face any challenges related to their background and experience. P4 stated that having a background in traditional development makes it easy to start using low-code development platforms, and lacking that was a problem for the citizen developers: *"I would say if you have a background in traditional development, then switching to low-code, or at least to OutSystems, is quite easy. We have seen people who started as citizen developers, and the problem they had is just that they do not know how to develop. They do not know how to structure stuff. The structure of a low-*

code program or a traditional program is mostly the same." (P4). The other participants (P2, P3, P5, and P6) –the citizen developers– reported some challenges related to not having experience or IT background, such as not knowing what is the best way to solve a problem and maintain high performance and maintainability. According to P2, *"A lot of challenges come from looking for the best way to do what I want. Most people can solve problems and come up with a way to follow through. Still, the hard part is making this performant and readable or maintainable for other developers"* (P2). Another challenge reported by P3 was having no grip of some software development concepts such as object-oriented programming. *"The hardest thing for me was the whole concept of object-oriented programming. So it helps if you understand already an object-oriented language like .Net or Java, then you start with Mendix, it goes way faster."* (P3). P5 talked about how difficult it is to learn low-code development when there are no low-code experts in the team who can assist the training, and the hands-on practices, *"None of us had worked with this platform before. There was a massive learning curve involved, where we had to train ourselves, and continuously we hit roadblocks of either not knowing how to do something, or if it was even possible."*(P5). Furthermore, P6 confirmed that it requires some hands-on experience and have a background in the development principles. *"Training courses are not enough. It would help if you had some hands-on practice. Challenges are mainly related to the development and design principles."* (P6).

5.1.2. Extensibility and Flexibility

Some LCDPs were found to constrain developers' abilities to solve problems and put a personal touch because of the limited choices, this imposes extra steps to solve problems and creates some technical debt Dahlberg [2020]. Most LCDPs have limitations that require the developer to do programming, and they might be more reliable when building minor automation or proof of concepts (PoCs). Furthermore, the lock-in (the dependency on the third-party platform owners) restricts the developers from having the flexibility they would like when developing a solution Virta [2018]. The paper of Tisi et al. [2019] highlights some limitations that hinder the use of LCDPs, such as being mainly addressed –from a scalability perspective– for building small apps. Their employment in building large-scale projects and mission-critical enterprise applications is not covered currently.

From the interviews, P1 did not face challenges related to this aspect. P1 thinks that OutSystems provides excellent options to extend applications and no need for any external development, *"I think this is the best part. So they have created so many connectors, components, and plugins so that you can extend your application. You can make use of them without having to develop a complete wrapper of your own."* (P1). On the other hand, participants P2, P4, and P6 faced some challenges when extending the solutions they built. P2 complained that OutSystems does not provide an integrated IDE with debugging or highlighting and IntelliSense features, making it very challenging to implement an external addition and use an external package. P2 had to use external tools which were not easy to use by a citizen developer. *"A client wanted a planning calendar, not something built into the application. We had to use a JavaScript library, and implementing that was a nightmare in OutSystems, because their scripting box is nothing more than a text editor, and you have to hope that you wrote it right the first time. You need a secondary tool most of the time. As a*

citizen developer, my troubleshooting or writing of this code was very much trial. OutSystems uses .Net and C sharp, so you have to install Visual Studio to write the C sharp code, while with Mendix, they have a built-in Java IDE with highlighting and IntelliSense, also the same for JavaScript. So that one helps facilitate the extensibility a little bit more. But downloading an external package is not always easy because of how the resource requirements work." (P2). Moreover, P4 had to learn a new programming language to build extensions because OutSystems supports only a limited set of programming languages, which was a challenge. P4 mentioned that OutSystems could connect to SOAP and REST services easily. *"The only challenge was to learn C sharp because I did not know that yet. So JavaScript, CSS, and C sharp for extensibility, really well manageable, and it also has excellent ways of connecting to existing SOAP and REST services, which is quite easy to do."* (P4). Another challenge reported by P6 was about adding external plugins to Appian, as there was no practical part about that in the provided training. *"Challenge with adding plugins that are not in the Appian App Market. The course does not give examples."* (P6). P3 did not deal with this aspect, however, P3 mentioned that building an extension in Mendix requires advanced development skills, *"I did not work on this part of the applications, but to connect external libraries to your application, you might need to build a connector or develop an interface between your application and the external systems. If you have to do it yourself from scratch, then you need to be an advanced developer most of the time."* (P3).

5.1.3. Code-review/Version Control and Collaboration

Low-code development is still a new approach, and it has no systematized techniques to perform its code review compared to the traditional code review Dijkink [2020]. In all LCDPs, the collaboration is mainly reflected between the developers and the testers Khorram et al. [2020]. According to Dahlberg [2020], teamwork, communication possibilities, and collaboration in low-code are challenging. Furthermore, some developers revealed negative feelings about collaboration when using Salesforce as there is no version control, which means no support for safely working together. There is a risk of overwriting each other's work. It is difficult to highlight changes or make them traceable, making it hard to track issues and do code reviews because of a poor overview of what others have done. From the findings of Khorram et al. [2020] and Ragusa and Henriques [2018], the centralized repository makes it challenging to understand the context of a specific change in OutSystems. The concept of a commit does not exist, and when developers are publishing changes simultaneously, an accumulation of unrelated changes is created, making the code review process impossible. There is no branching possibility to coordinate the teamwork, and performing pre-commit reviews are not possible because of using publish and commit system in OutSystems.

Three participants (P1, P2, and P5) confirmed facing challenges in this aspect. P1 stated that the version control in OutSystems is not comparable with the one of the traditional development that provides complete visibility of the code, but they are improving it. For example, it is possible now to compare the difference between the previous version and the new version, revert to another version of the software. P1 also mentioned that the support level of the version control depends on the low-code application platform, and some platforms are more mature than others. *"Yes, there are challenges, but the platforms are changing and gaining some maturity there. With the traditional development, we have the*

complete visibility of the code, but that is not the case with a low-code platform. You only see a few configurations and maybe a small piece of advanced code you have written yourself. We have seen the progress, you can compare the difference between the previous version and the new version, and the ability to revert to another branch or another version of the software. So things are going to that level, but there is still room for improvement. It also depends on the low-code application platform. For example, if OutSystems is at number nine on a scale of one to 10, Mendix could be at 10, or Power Apps could be at seven. So there are differences in the platform's maturity." (P1). According to P2, in OutSystems, there is only one branch to merge to, whereas in Mendix, it is possible to have feature branches, merge them later, and solve conflicts. OutSystems makes it possible to publish with one click, but if two developers do that simultaneously, one of them will need to wait, and there is a chance to overwrite each other's work. P2 also mentioned that in OutSystems, individuals have more responsibilities for code review as it is possible to click a button and add new features without informing the team. *"In OutSystems, there is only one branch, and you always merge to it. Other ones like Mendix have a traditional branching, and you can have feature branches, merge everything later and solve conflicts then. OutSystems simplifies the publishing process with one click; however, if I click the button at the same time as someone else, then one of us gets told you cannot do that yet, you need to wait, and you hope that you are not working on the same page, or in the same area of the application, or your function does not rely on anything that they are using. There is a lot more individual responsibility for code review with low-code, or at least with OutSystems because of their manner of how it works. For example, in traditional development, you can do pull requests, and that is not brought into your master branch until it gets approved, which almost forces a code review before it is brought in. With OutSystems, I can click that button and then not tell anyone, and they will not know this feature has been added. There is no built-in validation or verification or even necessity for code review, so it comes down to the development team."* (P2). In Alteryx, it was not possible to work simultaneously in one pipeline, according to P5. It has a primitive version control with only merging functionality and nothing else, which causes disturbance. *"Working together in one pipeline was not possible like other traditional development. It had the most rudimentary version control you could imagine, incrementing every push and no real checking. It is one branch only, and all the changes get pushed to the same branch. No conflict management, nothing to purchase. That gave us a lot of headaches."* (P5). On the other hand, P4 clarified that although OutSystems does not have the traditional version control tools, they did not face problems. *"It does not have the traditional source code tools like Git, so you need to be aware that someone else is developing in the same module. We have hundreds of modules, literally, but it is almost never that run into problems with us."* (P4). P6 also had no challenges related to this aspect when working with Appian. It is possible to switch between major versions, the team followed an agile approach and were linking the changes to Jira tickets to track them, besides the functionality of the automated versioning manager of the platform. *"We follow an agile approach to work with Appian. We have sessions to plan changes that are linked to JIRA tickets. Also, the automated versioning manager manages the files of Appian applications and databases in a version control system."* (P6).

P2 added that during the OutSystems training for citizen developers, there was nothing about code review, testing, or working with a team.*"When following OutSystems training, as a non-developer, you are not taught on code review or test, how to write tests, how to write*

unit tests. There is no instruction on how to work with a team. Okay, based exclusively on the training, my training was mostly individualistic. There is no indication that a code review is even useful, which can negatively affect the application's quality."(P2). On the other hand, Mendix supports the complete application lifecycle management. P3 clarified how that is done, from capturing the requirements to implementation, version control support, deployment, and test. The platform coordinates the team collaboration by providing the tools to design sprints, create user stories, tasks, and Scrum boards. P3 mentioned that Mendix has a massive difference from other low-code platforms in supporting this aspect. *"You have the support for the application lifecycle management, which starts with the customer and capturing the requirements, and building. You have those features or functionalities to deal with branches, merges of the models, handle the platform's extensibility with reusable items or components, and then one-click deployment. They support the full application lifecycle from deploying your application to test acceptance to production, and the involved business users provide the feedback that gets converted into a user story, and you start developing again. The full application lifecycle is supported and integrated, which is a massive difference from other platforms. You can design sprints, user stories, tasks, and story points, so your Scrum boards are digitized and part of the platform."* (P3).

5.1.4. Documentation and Support

In some LCDPs, it is not easy to get an overview of or information about the used assets due to limited documentation Dahlberg [2020]. The obscurity of the offered tools is another challenge. In Salesforce, for example, it is not always clear which tool should be used, as the tools overlap, such as Workflows, Flows, and Process Builders. It is not distinct what to use in a particular situation Virta [2018].

In this aspect, all the participants said they had no problems with the documentation or the platform support. P1 and P4 mentioned that some viral components are missing the correct documentation, but these components were created by the community, not the platform. In such a case, a traditional developer can look into the code and understand, but for a citizen developer, it would be a challenge. *"There are a few of the very popular components developed by the community missing the correct documentation in terms of using and maintaining the application. I would not blame the low-code application platform because you and I develop these."* (P1). *"Sometimes the components are not documented very well because community members created them. But it is open source. So you can just look into the code what it does if you want to know something. So I can imagine that someone who starts using OutSystems and is not an experienced developer might be a challenge if documentation is lacking."* (P4). Additionally, P1, P2, P3, and P4 clarified that everything was well documented and that both OutSystems and Mendix have forums where practitioners can reach out for help and solve questions. P3 stated *"everything is documented, videos and texts, and documentation. There is a forum where community members, including employees of Mendix, are on, also the people around the world helping each other with questions."* (P3). P1 and P4 are active members in the OutSystems forum, P1 as a champion and P4 as an MVP (Most Valuable Professional). According to P4, *"There is the forum where several active members reply to questions and answer questions. As an MVP, I do that, but there are many other MVPs and champions, as people with a lot of knowledge are very active on the forum and answering questions. There is an extensive library of videos to watch, explaining*

topics." (P4).

5.1.5. CUSTOMIZATION

According to Alamin et al. [2021], customization is one of the biggest challenges developers face when working with low-code platforms. Challenges concern adapting the work for a specific scenario, such as business logic implementation, input and form validation, linking the UI to the backend storage via dynamic content binding, a drop-down menu with predefined value, formatting date and time, and drop-down widgets. Customization could be in middleware (providing support for system integration, the connection between UI and storage layer) or the UI (drag-and-drop UI, form design, and customization of UI components).

Four participants (P1, P2, P3, and P6) confirmed having challenges related to customization. P1 stated that the citizen developers do not have enough skills to customize what the platform provides. *"There were challenges that citizen developers were not up to that low level that they can customize something already available and make it huge."* (P1). In some cases, it is challenging to implement a given workflow with low-code. According to P2, traditional development provides freedom and more capabilities, while in low-code there are boundaries and limited capabilities. *"One project in particular, where the client already had a particular way of working, and they did not want to change it. So there was a huge challenge in trying to match their way of working. That would have been much easier to implement in traditional development as you have much more freedom in a traditional way of working, and the capabilities are much greater. With low-code, it was a little bit of a struggle to get things exactly the way they wanted. You have to work within the boundaries of low-code, also coupled with a not-so-experienced team. Even though we had one or two experienced traditional developers, their knowledge does not always translate one to one over to low-code."* (P2). P3 stated that it is a challenge to build an external extension in Mendix, and using the platform for the wrong use case, such as gaming, is another challenge. *"If it is not supported by the platform natively, then you are dependent on extensions. If there is no extension available, you need to build it yourself, which could be a challenge. Other challenges are if you would like to use the platform for the wrong use case, for example, gaming."* (P3).

5.1.6. DATA STORAGE AND ACCESS

Alamin et al. [2021] reported some database challenges that low-code developers are facing, such as database connection, SQL CRUD operations, and import/export existing data. Data storage and access challenges can be in SQL CRUD (SQL query and table joining), Data Storage and Migration (uploading and storing files on the server, moving files from one platform to another, converting large CSV files to Excel sheets), or Entity Relationship Management.

Only P2 mentioned some challenges related to this aspect. Writing SQL queries is not easy for many people because SQL is a language on its own, and they need to learn it first. Another challenge is to build a good data model that best suits the handled use case, which requires some knowledge that citizen developers do not have. *"OutSystems has aggregates,*

a simple, straightforward way of creating an SQL query from the database. It is click-and-drag and all of this, but the moment you need to filter or join tables interestingly or oddly, you have to start writing your own SQL query. That is a struggle for many people because it is not something you can easily get. SQL is a language on its own, and you need someone willing to learn that too, to best fully handle all the scenarios, which is not always available in a low-code citizen developer team. We had a data model that did not match our use case, so we wrote a weird SQL query that caused an issue. The fact is that it is easy to set up a connection to a data source, whether it is a database or API in a low-code platform, but you want that to be performant. So having a simple query or a simple API call is easy, but the moment you need to consolidate a lot of data, it becomes challenging."

5.1.7. INTEGRATION

Integration is also one of the challenges when working with low-code, such as email server configuration, integration of external services, and OAuth fetching and parsing data. These challenges could be external web request processing (parsing or debugging the responses of REST APIs, some general query on networking protocols such as HTTP), or external API and email configuration (API integration, emails configuration error, using a generic programming language to send an email, creating managing calendar events) Alamin et al. [2021]. In some LCDPs, when options are limited, it is possible to use high code and integration. But it could make things more complex, and it might be better to move everything to code Virta [2018]. Salesforce's LCAP has no integration and API management support, and its process management is relatively limited. Both of these abilities are essential to modern application design, and delivery Vincent et al. [2020].

Most of the participants did not talk about challenges in this aspect. Only P6 has experienced some complexity in setting up and configuring integration with third-party systems. *"Integration is good, and you can communicate with third-party systems, but there are some complexities and not enough data available to learn and call integrations."* (P6)

5.1.8. TESTING

The lack of a standard framework resulted in the high dependency of existing LCDPs to technical third-party testing tools that are not usable for citizen developers. Additionally, although some LCDPs propose new low-code testing frameworks, they do not meet all low-code testing features, since they are not reusable for other LCDPs, and their resources are not open publicly Ragusa and Henriques [2018]. According to Khorram et al. [2020], four LCPDs (Mendix, Power Apps, OutSystems, Lightning) propose a new Low-Code Testing Framework (LCTF) but with restricted capacities. Consequently, they all afford integration with third-party testing tools to have good coverage of all testing activities. Functionality and performance are the continuously tested characteristics in all LCDPs, and they disregard testing other non-functional specifications. The citizen developer is only involved in the testing activities supported by LCTFs and when recording tools provide Automated UI testing. As an expert on the system functionalities, the citizen developer is accountable for defining the requirements. Consequently, she can create test cases and assess test results. Therefore, her engagement is essential in low-code testing, but her low level of technical experience creates many challenges; hence new techniques are needed.

None of the participants faced challenges in this aspect, but they gave some remarks. P1 and P4 stated that the OutSystems platform lacks testing tools, and it is possible to integrate external tools, but this is not easy. *"There is a lack of tools and libraries, which are not available in the low-code area compared to what we have been practicing so far. We have a few frameworks, which we can use, but of course not all."* (P1). Furthermore, P2, P4, P5, and P6 confirmed that testing is done manually and depends on knowing the application, its functionality, and proceeding through the possible scenarios. *"Automated testing is not provided out of the box. There are some testing tools that you could integrate with OutSystems, but it is not easy to do that. So it is manual testing and relies on the developers' knowledge and capabilities and the knowledge of the application. They will go through all the scenarios and see if everything functions as we want."* (P4). Mendix, on the other hand, supports unit tests, automated tests, and it has regression and consistency checks as P3 explained, *"What Mendix supports is like unit tests or application quality management, and lots of modules, as are part of the platform, which you can use to run test scripts automatically, start unit tests on top of the regression and consistency checks, which come with the platform. So the data model with the logic with the UI, all is tested on consistency. It is really hard to deploy an application that is not technically sound."* (P3). In Alteryx, the test was plain and based on having the right outcome, as P5 mentioned, *"I would basically always start with the outcome. If the outcome would make sense, then 99% sure that that would work well."* (P5).

5.1.9. Technical Debt and Refactoring

In her thesis, Dijkink [2020] concluded on the code smells and bad patterns that occur the most in low-code. The data for this study were collected using a quality analysis tool provided by OutSystems called Architecture Dashboard runs a collection of predefined rules to disclose certain code patterns. These code patterns are related to Performance, Architecture, Maintainability, and Security. Some of them are common such as Long undocumented flow, Avoid hard-coded literals, Lack of module classification, Cyclic references between applications, Large action, Duplicated code, Large module, and Switch statements.

From the interviews, only the traditional developers (P1 and P4) talked about this aspect. Neither of them faced related challenges. P1 explained that in traditional development, the code is visible, and it can be easily refactored. In low-code, it depends on how the platform has been designed or engineered. The development team needs to follow the vendor's recommendation to avoid creating technical debt. *"I am personally lean towards the standard way of software development because I have the complete visibility on the code, I can manage the classes, the complete interface, and I can design it the way I want, but with low-code applications, you have to think from the way basically how the platform has been designed has been engineered. So there is a different way of looking at the development practices. If you are not going to align with what the way software is recommending us to do, then you are basically going to create a lot of technical debt and insecure application."* (P1). From P4's point of view, it is easier to find technical debt in low-code as it is easy to inspect visual code, *"The nice thing about low-code is that it's easier to detect technical debt. On the one hand, it's easier to inspect visual code. On the other hand, you can easily see the program flow."* (P4).

5.1.10. Other Challenges

At the end of each interview, the participant was asked to mention any other challenges. Only P1 and P3 had something to add. P1 talked about criticism and adoption challenges that face low-code development. People think low-code means low quality and low capabilities, it can not deliver complex applications, and that low-code is for naïve or citizen developers only. *"With any new platform coming into the ecosystem, it has to face various challenges, criticism, and adoption challenges. The challenges were more on the non-technical side, behavioral side, and less on the technical side. When you start talking about low-code, people have this mindset and assumption that low-code means low quality, low level of application, low skill set. They think it cannot deliver the complex application or that low-code is for those developers who are not good at normal programming or citizen developers."* (P1).

P3 highlighted the challenges related to the extensibility and customization we listed before, especially for citizen developers, and described them as the main challenges when working with the Mendix platform. P3 also emphasized using the platform for the right use cases otherwise it does not make sense to use it. *"The main challenges you might have are when things are not supported natively by the platform, and you are dependent on the extensibility of the platform, and the components you want to use are not available in the app store or pilot through the community. Then it might be hard to develop them yourself if you have a non-technical background. So that is the main challenge; you need to use the low-code platforms, where they are suitable. So that is a wide range of transactional event-driven applications of all kinds of industries. No matter what complexity, performance, or scale you are in, if it is non-transactional or nonevent-driven, it does not make sense to use those platforms. Thus, most obstacles come when you use the platform for the wrong reason.* (P3).

5.1.11. Summary

In this section, we presented the answer to the first question in our research. We listed the reported challenges in the related work and the interviews and grouped them following the generic labels. In general, the findings from the interviews corresponded with and confirmed the ones from the related work.

5.2. RQ2: What impact could these challenges have on the software maintainability and performance?

Background and Experience

Performance and maintenance issues can arise with large systems when using the low-code tools, which could be associated with not understanding the software engineering principles Ness and Hansen [2020]. According to P1, some LCDPs have some inbuilt mechanisms that analyze and control the development output to improve the application quality, while in traditional development, the development team is responsible for that. This could be an

advantage of using low-code platforms. However, when the development team does not know how to use the tool, the performance and maintainability are compromised. *"You have a plus point using a low-code platform because if you build something using the traditional languages, you have to take care of the quality aspects. However, using these low-code application platforms, such as Power Apps, OutSystems, Mendix, or Salesforce, etc. These platforms provide you with an inbuilt mechanism to take control and analyze what you are developing. If you do not know how to use the tool, the performance or maintainability will be compromised, but not extensively."* (P1).

Documenting what you build is a vital step in software development, but citizen developers miss this part as it is assumed to be done automatically by the platform. P2 stated that this has a detrimental effect on the maintainability and extensibility of a low-code application. It makes it hard to find what caused a bug or to extend an existing feature. *"When working within a team and if there is ever a bug or an issue, a lot of this assumed good self-documenting proves to be detrimental, as it is hard to find the actual cause of a bug or it is hard to expand on the current feature. It is not always clear what is happening unless you deep dive into the functionality. In the traditional development, people know that it is necessary to write some documentation, and this is not always the case with low-code since it is assumed to do it for you."* (P2).

It is not wise to have a team of only citizen developers and ask them to build an application using a low-code platform. According to P3, they will face troubles, get stuck with any complex logic, and build poorly performing applications. *"If I were the only developer back then, I would have made mistakes, and the application would not have been as performant as it should have been. I would have got stuck somehow with complex logic, I guess. But yeah, the first six months, I was in a team with a more senior technical person."* (P3). For P5, lacking the experience caused refactoring the system three times, and the team needed six months of experience to set it up correctly. *"In the two years that I have worked with Alteryx, we have refactored the same transaction pipeline three times because of primarily hindsight. Half a year later, working with this low-code platform, we learned so much, and we can now set it up better."* (P5).

Extensibility and flexibility

In Virta [2018], the performance was the most significant issue reported in Salesforce, Some tools cannot handle larger data volumes and the complexity of the implemented systems. Commonly, the low-code solutions get converted to code because of performance issues, or some automation is taken out to make the system plainer. The low-code applications' performance is not comparable to hand-coded applications', especially with larger data volumes and complex systems Khorram et al. [2020].

P1 stated that when extending an application you can focus on the requirements and context, and all the non-functional aspects such as security, performance, privacy, etc. will be managed by the platform, however, the quality should not be ignored. *"In terms of these aspects, talking about the security, the performance, the privacy and a lot of other things, these are taken care of by the platform. That means, if you are extending your application, you can focus on the business context and requirements, not the non-functional require-*

ments outside of the business. You can always extend without worrying more about the quality aspect, but it is not something you can completely neglect." (P1).

P2 had a different opinion here, mentioning that the challenges in this aspect influence the maintainability and performance of applications, particularly when the citizen developers need to work with external libraries or extensions that are traditionally developed. *"That also influences the maintainability and performance of applications, especially when dealing with external libraries, and the team all are citizen developers because maintaining a traditionally developed package or a traditionally developed extension is an entirely different mindset, at least in my experience."* (P2). P2 added that some libraries or packages can not be used in the low cod platform because of configuration-related difficulties. this forces the low-code developers to build these extensions themselves, and this affects the maintainability. *"JavaScript libraries or front-end frameworks cannot always be used within the low-code platform because of the requirement to install resources or point to a particular NuGet package, and C sharp extension is not always available. So I feel like it lacks that last little bit to make it worthwhile for everyone to adopt. I think this also affects the maintainability because developers are forced to write this functionality themselves. And again, as citizen developers with no traditional development background, that affects the maintainability of their extensions."* (P2). P4 confirmed what P2 said regarding the maintainability, however, P4 added that the performance is not affected in this aspect. *"It does not affect performance. But, the maintainability is lower because it is a traditional code, so it is less easy to manage. We are limited to the really necessary things, but we do everything else in the low-code platform."* (P4).

Code-review/Version Control and Collaboration

Adopting a post-commit review in OutSystems enables developers to continuously commit changes to the repository while other team members can review the code changes and alter their work, respectively. However, this approach will affect the quality especially the maintainability and the performance as more poor code will find its way to the central repository Khorram et al. [2020]. Furthermore, adding comments inside the low-code developed applications in Salesforce is impossible, which makes the maintenance of the code even harder Dahlberg [2020].

Code-review practice plays a crucial role in optimizing both the maintainability and performance of applications. According to P2, it remediates the worry and risk of having bugs and improves the performance by considering the best approach to follow. However, this is missing in low-code development. *"If I write something and overlook a bug, my whole application reliability is shot. Having another pair of eyes look over the code helps alleviate some of that worry and risk, then the performance, because I might think one way is better or more performance than what you were working on. That does not mean the way I wrote it gives the most performance. It might be a combination of the two or something along those lines. And maintainability, having the other pair of eyes read it, they can see if it is maintainable enough just by reading through the code and seeing if they understand what is going on. That is never the case in low-code in my experience."* (P2).

Customization

In complex applications or when some functionality does not fully meet the requirements, then some customization is required. According to P1, when building a complex application with complex logic that leads to customizing a big deal of the application, the maintainability will become low together with other aspects, such as reliability, serviceability, usability, and so on. It also has to do with how well-structured the logic and architecture are of the application. *"They are less maintenance, but again, when you have a very large complex application, very complex logic, you have built that means you have a lot of custom code, many things which are together, for example, reliability of libraries, serviceability, usability, a lot of components basically to look at. That means you have to take care of these things on your own, these are not produced by the platform itself. It purely depends on how nicely well-structured your business logic and architecture."* (P1).

P2 assured that customization in low-code applications has a detrimental impact on maintainability and performance, especially when trying to reflect a complex workflow or logic on low-code. This leads to creating "spaghetti" code, which is not maintainable and costs time and effort. P2 mentioned that this would be straightforward with traditional development: *"This kind of customization makes you sacrifice the maintainability and performance of the application. People always joke about spaghetti code, and this was it. It was nuts just to try and get it to be exactly what the client wanted. It was not really maintainable because a lot of custom logic was written and was not written in the way that low-code is meant for. We end up with a lot more bulk, and you cannot streamline every function or method. You need to combine a few things to get to the same functionality that you could do straightforward with a simple JavaScript library or .Net library elsewhere."* (P2)

Datastore and access

In this aspect, the challenges that happen when querying the data influence mainly the performance. P2 talked about writing a complex query with the support of an SQL expert in the team, however, the performance was badly affected by that complex query. *"There was a SQL query that I had to write myself, and I got our SQL expert within our team from the traditional development world to support me on this because it was necessary. But the performance was absolutely dreadful. I remember it being complex and needing to take the results from one table, join them to a filtered table, or results from another table without affecting the original table. And what would normally be a non-issue ended up being terribly slow."* (P2).

Input about quality of low-code applications

P3 gave input about the quality of low-code applications he/she developed in Mendix. They have been developed rapidly and they are maintainable, extendible, sustainable, and robust from P3's experience. Most of the applications that P3 had developed between 2008 and 2012 are still in production and have already gone through some upgrades, extensions, and maintenance. *"Applications that have been developed with low-code, they were developed at a way faster pace than they used to. I have developed the most applications in my life between 2008 and 2012. We are now in 2021, and almost all the applications I have developed are still in production. Although they are upgraded into newer versions of Mendix, obviously, and extended, adding new features and maintaining them, and so on, they are still alive after all those years. So they are sustainable and robust. I think maybe that says*

enough." (P3).

Summary
In this section, we presented the answer to the second question in our research. How some challenges could impact the maintainability and performance of low-code applications; this was identified based on the data from the related work and the interviews.

5.3. RQ3: How does the development team deal with these challenges to reduce their impact?

Background and Experience
The team may be able to avoid performance issues by designing the system well from the start, which requires that the team is familiar with the best practices of software engineering Ness and Hansen [2020]. The challenges in this aspect were reported by the citizen developers only. The participants talked about some practices and actions they followed to overcome these challenges and improve the quality of their work both on the maintainability and performance sides. Self-educating, for example, to learn about the conventions of traditional development and the design principles as P2 stated: *"I did self-education, as it is still software, so I wanted to make sure that the conventions occurring within traditional development are still being followed within low-code. Also, it is good to learn about design principles to help you build and maintain a maintainable application and good performance."* (P2).

To ensure that citizen developers document their work, this should be communicated clearly and added to the default practices and way of working. *"It needs to be brought to the attention of the developers that an extra step is needed. Maybe a description is written for something or an additional comment added. So a change in the mentality of the developers."* (P2). According to P3, one of the most important requirements to build a good quality application is to have some senior developers in the team to guide the development process and control the delivery. Furthermore, the citizen developers need to gain some experience before delivering real work. *"Having some senior people in the team is kind of mandatory to deliver good quality applications. It is necessary. You can still make mistakes, which affects the application's performance and maintainability, and I think you need to build up some experience before levering."* (P3). P5 mentioned using two ways to resolve the challenges they faced, one was using Google, as the platform has good community and documentation. The other way was that they bought outright through a Dutch particular reseller who has a very well-staffed Support Center desk to help to fix any roadblocks they face. Lastly, what P6 suggested is to let the citizen developer go through the documentation during the training and then do a month of hands-on experience using the platform. *"I would suggest that along with the training courses, you should also go through the Appian Documentation and have at least a month's hands-on experience of the product."* (P6).

Extensibility and Customization
To make building extensions smoother, P2 used an external IDE. *"I use Visual Studio Code as a secondary tool to support myself as an external IDE"* (P2). Furthermore, P3 stated that

building extensions should be assigned to traditional developers. *"If you have to do it yourself from scratch, then yeah, you need to be an advanced developer most of the time."* (P3). On the other hand, the policy P4 followed was to only build extensions when there is no other option. *"So we are limited to the things that are really really necessary"* (P4). P4 also mentioned that when an extension or customization is needed, then the first place to look for it is the OutSystems Forge (a container of reusable, open-source components, modules, connectors, and custom solutions), as someone could have solved the problem. *"If you are looking for something, the first place to do would be to look into OutSystems forge and see if someone else has already solved your problem, which is very often the case."* (P4)

Code-review/Version Control and Collaboration

In Dahlberg [2020], a developer stated that the best way to work is by having a small team, talking to each other often, making sure to work in separate parts of the system as well having a well-defined data model from the beginning. Collaboration within a team can be improved by following some best practices and strategies that help to pass the challenges in this aspect as P1 stated. These practices can be set by the team or provided by the platform vendor. *"All the applications were developed within a team that was not very big. We followed the best practices and strategies for better collaboration and to pass these challenges. We did not face any real big issues. We need to educate ourselves better, and we have to follow the guidelines, which the vendor has already written down to avoid any mistake and improve the collaboration"* (P1).

P2 also gave an example of how to improve collaboration by following an agile framework such as Scrum. *"I work in a team with many traditional developers who follow Scrum, which supports the collaboration and communication efforts in its way because we use external boards, such as Azure, DevOps, or JIRA, to help consolidate and collaborate. "* (P2). To avoid overwriting each other's work, P2 mentioned that they used a communication channel such as Slack or Teams to coordinate. The code review is facilitated outside the low-code platform and it should be in the team's culture. *"There is a lot of communication required. So, we have worked with some teams who were operating remotely, and it is almost impossible unless you have Slack or Teams or something where you can say, I am publishing now. We have worked together as developers to say, hey, can you look over this? This is facilitated outside the low-code platform and required as part of the team's culture."* (P2).

As a strategy to avoid conflicts, P4 said that they try to separate functionality in different modules, and during the sprint planning the work is divided on the team based on the modules. In case any conflict happens, this is discussed and solved by the two responsible developers. P4:*"We try to separate functionally in different modules, so when we plan a sprint because we use agile Scrum, you already know what use case you will use, modules, etc. So then you already plan a bit, you are going to do that, then you can also do this because it is in the same module, something like that. And if they are really incompatible, then you would just talk to your colleague and say, hey, how would we resolve this?"* (P4). P4 talked also about a strategy to facilitate the code review. This is done by adding comments or visual commands to descript the change and add the initials of the developer, the date, and the use case number, which makes it easy to track the changes. The platform itself provides a way to compare the published versions, as P4 explained. Each published version has the

name of the publisher attached, so the changes can be extracted by comparing an older version with a newer one. *"As part of the normal sprint, we will always have reviews and code reviews of each other's work, and this is quite easy because we have decided to mark everything with comments and visual commands. So just like yellow squares, you can put a comment and everything that is changed in a module. We put our initials in the date and the use case number. So you can always trace back what has changed in the code. Also If you publish a version, then your name will be attached to that version. So if I see there was a change, and I do not know exactly who made it, then I can compare an older version with a newer one. And then the tooling will say, Okay, these are the changes, these are the differences between those two."* (P4).

Having the expert in the team to do the code review, is the strategy that P5's team was following. P5 was the expert and was responsible for doing the code review in the team. This was done by looking together with the developer to the code. *"The code review was fully done by me. Once we had a process in place where the analyst would finalize their codes, I would look together with them if it would make sense and if the result indeed produced the things we expect. That was our strategy as I was the most knowledgeable on this platform."* (P5).

Datastore and access

P2 proposed a way to resolve the challenge they faced with a complicated query that caused slow performance. The solution would be to adjust the data model and create another table or anything that could help to optimize the query: *" So it would have probably been better to rework the data model or create another table or something like that."* (P2).

Testing

The testing aspect and cooperation between the developers and the testers in low-code development can be improved according to P2, and this can start by creating well-documented acceptance criteria. The tester will have an idea of where to focus and what is needed. The citizen developers should learn about this. *"I will say one area that needs to be facilitated a little better is the testing area and cooperation between the developers and the testers themselves. Because in traditional development, the acceptance criteria are very well documented. This isn't always the case in low-code, because again, coming from the world of non-traditional developers, a lot of times the result is okay, the feature is made, it is ready for a test without a lot of the extra information that would be useful for a tester on where to focus or what is needed for this to be a success story."* (P2).

P5 described how they were testing their work and what strategy they followed. The analysts were responsible for testing their work and controlling the quality. They assigned someone to test the final results before the deployment, and P5, as the expert in the team, was monitoring the development process. *"We gave our analyst full responsibility for the quality of the end product. So we basically pushed them to test by themselves extensively. And then also have somebody else test the end results before really deploying. I was also usually involved during the development process to make that kind of control, so that is the strategy."* (P5).

Technical debt and refactoring
P1and P4 talked about the AI-based advice OutSystems platform provides about the architectural state and implementation of the low-code application. This helps to refactor where needed and resolve most of the technical debt. P1 added that Mendix and some other LCDPs have something similar: *"OutSystems has some tooling to inspect the architectural state of your applications. So you can run that, and it will do code analysis, and it will analyze how the different modules are connected. And then it can give some advice about what you need to change, architecturally. So that's very helpful as well."* (P1).

Adoption challenges
Concerning the adoption challenges that were mentioned under the generic label "Other Challenges", some participants stated their opinions about low-code development clearly. P1 thinks it is not about what you are using to develop but more about how you develop your design or application and whether the software development principles and best practices are followed. *"I have a different opinion there. So it is not that the application or the language or framework is bad or good. It is how you develop your design or your application. It is the complete software development capacity, what are the laws, the best practices if you are not going to enforce the best practices then you are basically going to create a lot of technical debt and insecure applications."* (P1).

P2 expressed that low-code development platforms are much better if they are used principally by traditional developers who follow the best practices of software development because these platforms assure to have most of the quality aspects covered at a reasonable level. *"These low-code platforms are starting to target the traditional developers, C#, .Net, or Java developers, to show how it can expedite their development process speed. I think this is great because they can bring the ways of working from the traditional development such as the documentation of code, keeping the maintainability high, writing or creating the models, making the logic flows in a maintainable way, and following the design principles and development frameworks. I think that will greatly improve the overall maintainability because, in my experience, the minimal security, minimal reliability, and minimal maintainability are all higher than that of writing something with traditional code. There is no hand-holding when you write a Python back end, for example, or a JavaScript front end. You can do whatever you want. You can leave all the security holes you want with no checks. You might get an error here or there or a warning, but you can still publish that and put it on the web, while with low-code, at least a lot of those holes are covered. They might not be the strongest cover, but at least they are covered."* (P2).

A lot of traditional developers disfavor low-code development according to P4. They think it is only for citizen developers, which is inaccurate. P4 thinks that citizen developers cannot develop well and they only increase the problems. P4 added that low-code accelerates the development, makes code inspection easier, and improves the performance and maintainability of applications. *"A lot of traditional developers look down on low-code development. I can imagine that if you have no code development that is really targeted towards citizen developers, which is a bit of a misnomer, citizens almost can never develop, just doing stuff but creating more problems than needed. But in general, low-code is just a way of faster developing the same code, and you still write code. It's just that you do not write all*

the boilerplate that you need to write a few doing traditional code development. So you can develop faster, and it is easier to inspect code written by others because it is more visual. So, in general, I think low-code is a very good way of increasing both performance and maintainability and software quality in general." (P4).

Summary
This section presented the answer to the third and last question in our research. We investigated how low-code developers dealt with the challenges they faced to decrease their impact on the maintainability and performance of applications they built. The data from the interviews were largely used in formulating the answer. We have seen how development teams tried to identify some best practices and put strategies to help them bypass any roadblocks.

5.4. Recommendations in Low-code Practice

We asked the participants to share advice or some recommendations for the low-code practitioners or platform vendors. Here we list their answers.

Low-code practitioners P1 emphasized following the guidelines and best practices that the vendors share, and to think about the proper data model and architecture. *"Please follow the guidelines and the best practices which are being developed, published, and updated. They are updating quite often with every new release, follow their best practices. The platform provides a lot of capacity and the ability to maintain the application and the good quality. If you do not follow this, you will create a spaghetti, a non-maintainable system, which has a low-quality aspect, which we want to avoid. Low-code does not mean just drag and drop and building applications. When you are building something, think about the proper data model and the relationship between the entities. Do they do their work on paper? Try to understand what you are building and the best architecture for it."* (P1). P2's advice for the citizen developers is to learn about the software development principles, design patterns, and best practices to apply on low-code. *"Do some external research about design patterns and principles and all traditional development best practices to help frame how to approach a problem or a challenge within low-code."* (P2). P3 Mentioned some external companies that can provide quality scans and label on low-code applications, such as Omnext and Software Improvement Group, and doing this research at KPMG Digital we know that the software quality team provides these quality reviews on both low-code and traditional applications. *"There are external companies who can provide a quality label on to the low-code application. Omnext and Software Improvement Group made it like the five-star ratings and various levels like modularity, complexity, and those kinds of things. However, they come with a price."* (P3). P4 advised the low-code developers to know the tool well.

Low-code Vendors P2 asked the vendors to facilitate the DevOps process especially the code review and testing activities from writing test cases to testing requirements. *"For low-code vendors: I hope they facilitate these DevOps processes, which involve*

code-review and testing and these areas. I know Mendix do it a little bit by integrating the DevOps board and environment into their IDE, but do go a step further, like help in writing test cases or adding the requirement for testing stories and these kinds of things that really help improve the quality and guaranty even higher level of it." (P2). P3 gave an example of how Mendix supports different types of developers by providing different environments, which is something other low-code vendors should consider. *"Mendix distinguishes three types of developers. The citizen developers access the Mendix no-code environment, and then the business engineers access the Mendix desktop environment. Lastly, the hardcore coders who are in the extensions of the platform."* (P3). P4 thinks that OutSystems focuses more on the citizen developers, so they should give more attention to the experienced developers as well. *"The general things like know your tooling, and know everything that you can do. The platform vendor should, in general, not forget about the experienced developers. Nowadays, many additions to the platform are geared towards citizen developers or towards people starting to develop, while many experienced developers want to get things done quickly and do not want to wait, it is aided with all kinds of nice things, for starters, but that is the only thing. They should also have focused on experienced developers."* (P4). P6 suggested improving some of the services that the Appian provides in their platform and expanding them. *"As an Appian developer, I have a few suggestions. Some smart services are not working correctly, or they have limitations; for example, Copy Document smart service is not working for Xls format, and there is no alternate for that. So for these kinds of limitations, there should be some solution or if it can be improved."* (P6).

6

Discussion

By analyzing the data we collected, we gain insight into the challenges the development team faces when working with low-code platforms. We investigated which challenges impact the maintainability and performance and how to reduce that. All three data sources helped in investigating the research questions and answering them. Reviewing the literature and the technical reports gave an idea about the challenges studies and market research reported and helped in knowing the state of the art of low-code. We derived the generic labels and preliminary results from both the related literature/technical reports and the low-code online communities, then we used them as input for the interviews. The interviews helped in drawing out the final results for all three questions.

6.1. Maintainability and Performance of Low-code Applications

From the results and the answer to our second research question, we can summarize that the maintainability and performance of low-code applications could get impacted when:

a) The application is large, complex, or implements not a supported use case.

b) The team is mainly citizen developers.

c) Extending the application with traditional code.

d) No code review, amber version control, and not following a good development framework methodology.

e) Customizing an existing workflow or component to implement complex logic.

f) Creating an inconsistent data model/architecture and writing complex data queries.

Some LCDPs have some inbuilt mechanisms that analyze the development output quality, which is an advantage of using low-code platforms. However, if the development team does not know the tool well or the development principles, the performance and maintainability are compromised (P1).

6.2. Implications

Our findings have implications for the development teams and the practices they follow, as well as the design and characterization of LCDPs.

6.2.1. Citizen Developers

The citizen developers participants (P2, P3, P5, and P6) reported challenges in the Experience and Background area. It is much easier for traditional developers to use LCDPs, in contrast to citizen developers. P2 talked about educating citizen developers and bringing the development principles to their attention. On the other hand, P4 stated that citizen developers cannot develop, and they only increase the problems. Additionally, both P3 and P5 implied taking six months to have a sufficient level of knowledge to be productive and implement properly. The same applies to the Extensibility and Flexibility area. The citizen developers faced challenges when using traditional development technologies to extend applications. However, educating them and teaching them about software development and design principles could help eliminate these challenges. Unfortunately, the training they follow lacks even the practical examples, as P2 and P6 stated. Another area to improve is creating well-documented acceptance criteria, which are vital to enhance the cooperation between developers and testers and give an idea of where to focus and what is needed (P2).

6.2.2. Traditional Developers

Traditional developers can switch to low-code development easily (P4). They also work what requires high development skills, such as building extensions, writing custom SQL queries, or doing customization to implement complex logic (P2, P3). However, traditional developers tend to not join a low-code development team because of the adoption challenge (P1, P2, P4).

Both P2 and P4 think the low-code platform should focus more on the traditional developers. In their opinion, having the traditional developers' experience and the low-code tools and services makes the best combination to build high-quality applications.

6.2.3. Low-code Platforms

The challenges we found from exploring the literature were confirmed by the participants, except in the documentation and support area. Most of the users did not face challenges in this area, and this could be because the LCDPs we studied are different; the literature was studying Salesforce, while the participants in this research work with OutSystems, Mendix, Alteryx, and Appian.

LCDPs support extensibility and the extensions that do not exist in the related marketplace can be built using the same technologies used in the platform. Mendix uses Java, OutSystems uses C#, and both support JavaScript too. However, OutSystems does not provide an integrated IDE with debugging or highlighting and IntelliSense features, so it is difficult to implement an external addition or use an external package (P2). In such a situation, an external IDE can be utilized, for example, Visual Studio. Also building extensions in Mendix

requires advanced development skills(P3).

In low-code, there are boundaries and limited capabilities. The limited options in some LCDPs, force extra steps to solve problems and create technical debt (P2). Some low-code platforms support version control more than others. Mendix has the best support for this functionality (P3, P1). OutSystems provides one branch to merge to, whereas in Mendix, it is possible to have feature branches, merge them later, and solve conflicts. In OutSystems, the code review is not applicable (P2). The version control in OutSystems is not comparable with the one of the traditional development, but they are improving it (P1). Working in low-code could be more efficient as small teams in separate parts of the system and have a well-defined data model from the beginning (P2). To pursue the code review and track changes, comments could be added to explain the change and add the developer's initials, the date, and the use case number (P4). The testing is manual, and the functionality and performance are the continuously tested features (P2). Mendix supports unit tests, automated tests, and it has regression and consistency checks (P3). OutSystems and Mendix platforms provide AI-based advice on the architectural and implementation state of the low-code application, which could help to resolve most of the technical debt and improve the development (P1, P3). Most participants talked positively about the low-code platforms they used. P1 was the most low-code-supportive one and kept mentioning the improvement in this field.

6.3. Threats to Validity

When reviewing the related work there were not enough studies related to the topic of our research. We also limited ourselves to two technical reports of two independent research organizations (Gartner and Forrester). Furthermore, the discussions of the online communities were mainly about asking for help to solve issues while building applications, and not much about the other development aspects such as code-review, testing, collaboration.

We explored two low-code forums only and looked into a small part of what is being discussed, which impacts the results we got. Moreover, a threat to the external validity of this research is the fact that the participants worked with a limited set of LCDPs (OutSystems, Mendix, Appian, Alteryx). They might not be representative of all low-code developers as we know there are hundreds of LCDPs, and their input to this study might be biased. Additionally, there were only two participants who worked with more than one low-code platform (P1 and P2).

Another threat to the validity of this study is that we did not cover all challenges that hinder the low-code development process. The goal of this thesis is not to provide a comprehensive list of low-code challenges, but to highlight the most important ones and are interesting for further research.

6.4. Recommendations for Future Work

The contributions of this research are: providing insight into the challenges in the low-code development process, their impact on maintainability and performance, and how that is treated by development teams. Since this is an exploratory study, we believe our research

triggers some questions for future research. There are hundreds of low-code platforms, and we only explored some of them. It is recommended to research other platforms in future research to see if our findings are valid with other low-code platforms. Another topic to be investigated is the role of citizen developers in low-code, and how it can be improved. For example, what are the skills they need to learn to be able to customize and extend solutions? what design principles could help them to create robust architectures? Furthermore, the code review practice is crucial for software quality. To apply this practice in low-code development, a clear approach should be defined and potentially be shared between low-code platforms. Future work can try to frame this approach, apply it, and measure the results. Finally, each label we used in listing our findings could be considered as a topic for further research.

7

CONCLUSION

The study is concerned with investigating what challenges the development team faces when working with low-code platforms. Moreover, we tried to analyze their impact on the maintainability and performance aspects and describe how that is addressed. We followed an exploratory approach and went through three phases to extract the findings. First, we explored the related literature and technical reports and then checked the main discussed topics in two low-code online forums. Finally, we conducted interviews with low-code developers to get insight into the challenges they identify and validate the results from the other data sources. The results showed the challenges low-code developers are facing when working with low-code platforms and related to Background and Experience, Extensibility and Flexibility, Code-review/Version-control and Collaboration, Documentation and Support, Customization, Data Storage and Access, Integration, Testing, Adoption, and Technical debt and refactoring. In the end, we included some recommendations from the participants to both low-code developers and vendors.

This study gives an insight into the challenges in low-code development and suggestions on reducing their impact on maintainability and performance. It raises some important questions for further research in his field. For example, investigating the citizen developer role in low-code development and how it can be improved, or investigating how to apply the code-review practice in low-code development.

BIBLIOGRAPHY

Ritu Agarwal, Jayesh Prasad, Mohan Tanniru, and John Lynch. Risks of rapid application development. *Commun. ACM*, 43:1, 11 2000. doi: 10.1145/352515.352516. 5

Md. Abdullah Al Alamin, Sanjay Malakar, Gias Uddin, Sadia Afroz, Tameem Bin Haider, and Anindya Iqbal. An empirical study of developer discussions on low-code software development challenges. *CoRR*, abs/2103.11429, 2021. URL https://arxiv.org/abs/2103.11429. 29, 30

Hilary Berger, Paul Beynon-Davies, and Pat Cleary. The utility of a rapid application development (RAD) approach for a large complex information systems development. In Timo Leino, Timo Saarinen, and Stefan Klein, editors, *Proceedings of the 13th European Conference on Information Systems, The European IS Profession in the Global Networking Environment, ECIS 2004, Turku, Finland, June 14-16, 2004*, pages 220–227, 2004. URL http://aisel.aisnet.org/ecis2004/7. 3

Erik Blair. A reflexive exploration of two qualitative data coding techniques. *Journal of Methods and Measurement in the Social Sciences*, 6:14–29, 01 2015. doi: 10.2458/jmm.v6i1.18772. 20

Virginia Braun and Victoria Clarke. Using thematic analysis in psychology. *Qualitative Research in Psychology*, 3:77–101, January 2006. doi: 10.1191/1478088706qp063oa. 20

Daniel Dahlberg. Developer experience of a low-code platform: an exploratory study. Master's thesis, Umeå University, Department of Informatics, 2020. URL http://umu.diva-portal.org/smash/record.jsf?pid=diva2:1485348&dswid=-2690. 6, 9, 24, 25, 26, 28, 34, 37

Orla Dijkink. Occurrence of code smells in low-code. *Open University of the Netherlands*, March 2020. 5, 11, 26, 31

Brian Fitzgerald. A preliminary investigation of rapid application development in practice. In *Methodologies for developing and managing emerging technology based information systems*, pages 77–87. Springer, 1999. 3

Martin Fowler. *Refactoring: improving the design of existing code*. Addison-Wesley Professional, 2018. 11

Chris Grams. How much time do developers spend actually writing code? October 2019. URL https://thenewstack.io/how-much-time-do-developers-spend-actually-writing-code/. 4

Fadhl Hujainah, Rohani Binti Abu Bakar, Basheer Al-haimi, and Mansoor Abdullateef Abdulgabber. Stakeholder quantification and prioritisation research: A systematic literature review. *Information and Software Technology*, 102:85 – 99, 2018. ISSN 0950-5849.

doi: https://doi.org/10.1016/j.infsof.2018.05.008. URL http://www.sciencedirect.com/science/article/pii/S0950584917302422. 3

Stacy A Jacob and S Paige Furgerson. Writing interview protocols and conducting interviews: tips for students new to the field of qualitative research. *Qualitative Report*, 17:6, 2012. 19

Faezeh Khorram, Jean-Marie Mottu, and Gerson Sunyé. Challenges & Opportunities in Low-Code Testing. In *ACM/IEEE 23rd International Conference on Model Driven Engineering Languages and Systems (MODELS '20 Companion)*, Virtual, Canada, October 2020. doi: 10.1145/3417990.3420204. URL https://hal.archives-ouvertes.fr/hal-02946812. 5, 10, 26, 30, 33, 34

Ilker Koksal. The rise of low-code app development. April 2020. URL https://www.forbes.com/sites/ilkerkoksal/2020/04/29/the-rise-of-low-code-app-development/?sh=24a37a566807. 6

Mike McCormick. Waterfall vs. agile methodology. *MPCS, N/A*, 2012. 3

Alok Mishra and Ziadoon Otaiwi. Devops and software quality: A systematic mapping. *Computer Science Review*, 38:100308, 2020. 4

C. Ness and Marita Hansen. Potential of low-code in the healthcare sector. 2020. 6, 9, 32, 36

G. Ragusa and H. Henriques. Code review tool for Visual Programming Languages. In *2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 287–288, October 2018. doi: 10.1109/VLHCC.2018.8506527. ISSN: 1943-6106. 26, 30

Clay Richardson, John R Rymer, Christopher Mines, Alex Cullen, and Dominique Whittaker. New development platforms emerge for customer-facing applications, 2014. URL https://www.forrester.com/report/NewDevelopmentPlatformsEmergeForCustomerFacingApplications/-/E-RES113411. 4, 5

A. Sahay, A. Indamutsa, D. Di Ruscio, and A. Pierantonio. Supporting the understanding and comparison of low-code development platforms. In *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 171–178, 2020. doi: 10.1109/SEAA51224.2020.00036. 9

Raquel Sanchis, García-Perales, Francisco Fraile, and Poler. Low-Code as Enabler of Digital Transformation in Manufacturing Industry. *Applied Sciences*, 10:12, 2019. doi: 10.3390/app10010012. 6, 8

N. Sattar. Selection of low-code platforms based on organization and application type. LUT University, School of Engineering Science, Computer Science, 2018. URL http://urn.fi/URN:NBN:fi-fe2018080233306. 6, 9

Melania Sulak. Low-code vs. traditional development. May 2020. URL https://www.objectivity.co.uk/blog/low-code-traditional-development/. 4

Sebastiaan Tiemens, Alex Brouwer, and Swatantra Kumar. Low-code: Empower the capability to accelerate. *Compact*, March 2019. URL https://www.compact.nl/articles/low-code-empower-the-capability-to-accelerate/. 8

Massimo Tisi, Jean-Marie Mottu, Dimitrios S. Kolovos, Juan De Lara, Esther M Guerra, Davide Di Ruscio, Alfonso Pierantonio, and Manuel Wimmer. Lowcomote: Training the Next Generation of Experts in Scalable Low-Code Engineering Platforms. In *STAF 2019 Co-Located Events Joint Proceedings: 1st Junior Researcher Community Event, 2nd International Workshop on Model-Driven Engineering for Design-Runtime Interaction in Complex Systems, and 1st Research Project Showcase Workshop co-located with Software Technologies: Applications and Foundations (STAF 2019)*, CEUR Workshop Proceedings (CEUR-WS.org), Eindhoven, Netherlands, July 2019. URL https://hal.archives-ouvertes.fr/hal-02363416. 4, 25

Erik Vikebø. An Inquiry into Low-Code Solutions in Institutions for Higher Education. NORWEGIAN SCHOOL OF ECONOMICS, 2020. URL https://hdl.handle.net/11250/2644682. 10, 24

Paul Vincent, Yefim Natis, Kimihiko Iijima, Jason Wong, Saikat Ray, Akash Jain, and Adrian Leow. Magic quadrant for enterprise low-code application platforms, September 2020. URL https://www.gartner.com/doc/reprints?id=1-24TEFBCB&ct=201214&st=sb. 5, 13, 24, 30

Tatu Virta. Relation of low-code development to standard software development: Case Biit Oy. 2018. 9, 24, 25, 28, 30, 33

Robert Waszkowski. Low-code platform for automating business processes in manufacturing. *IFAC-PapersOnLine*, 52(10):376 – 381, 2019. ISSN 2405-8963. doi: https://doi.org/10.1016/j.ifacol.2019.10.060. URL http://www.sciencedirect.com/science/article/pii/S2405896319309152. 8

Lars Öbrand, Nils-Petter Augustsson, Lars Mathiassen, and Jonny Holmström. The interstitiality of it risk: An inquiry into information systems development practices. *Information Systems Journal*, 29(1):97–118, 2019. doi: https://doi.org/10.1111/isj.12178. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/isj.12178. 3

APPENDICES

.1. APPENDIX 1

Interview Script

.1.1. PREFACE

- Introduce myself
- Information is confidential. - Interview will last about half an hour
- Confirm receiving the signed consent form
- Content + walk through the main topics
- Recording and access
- Confidentiality + anonymity
- Any questions? Is everything clear?
- Ask some questions related to the background and career
- Confirm and start recording

Aim 1 Insight into what challenges low-code developers encounter while working in low-code development (process, activities, practices ...)

Aim 2 How these challenges could influence the software quality (maintainability, performance) and how that could be avoided or resolved?

.1.2. BACKGROUND OF PARTICIPANT

- Can you tell me something about your professional background, and where are you working/for how long? (not included in the recording)
- Can you tell me something about your studies and programming background?
- What is your experience with Low-code development?

.1.3. CHALLENGES IN LOW-CODE DEVELOPMENT PROCESS

- Can you tell me about the main Low-code projects you worked on? (Project Type, complexity, role, successful)
- About the low-code development team? (Size, roles, development methodology)

*** For each of the following focus areas, could you answer these three questions in case they are applicable?**

A - What challenges have you (or any team member) faced when working with low-code in this aspect?

B - Do you think the quality of the applications (maintainability, performance) could have been affected by these challenges? How?

C- What do you (or the team) do to mitigate the influence of these challenges on the application quality?

Focus areas:
- Experience (as a citizen developer/ no software development experience/ working with low-code)
- Extensibility and flexibility (of applications / how easy/complicated to extend)
- Collaboration (within the team/ coordinating/ development methodology)
- Code-review/Version Control (tracking changes)
- Documentation (good support/ easy to understand/ everything is clear how to use)
- Customization (how easy to customize to implement complicated logic/ adjust components/ using scripting languages)
- Database (design data model/ connection/ external DB/ querying)
- Integration (with APIs/ external systems/ difficulties/ support)
- Testing (tools/ frameworks/ best practices)
- Technical debt (popular patterns)
- If you have come across challenges in other aspects, please tell about them.

.1.4. Impact on Maintainability and Performance
- How do you consider the quality of the projects you worked on? (High/Low, performance, maintainability)
- Do you think the quality of the applications could have been affected by the technical challenges? How? (for each focus area) (Low performance, complex solution, low maintainability)

.1.5. Actions to Mitigate Challenges
- What do you do to mitigate any destructive influence of the challenges on the application maintainability/performance for each focus area? (Training, using analysis tool, best practices)

- Any recommendations for improvement in low-code development?
(Vendors to improve low-code platforms, Practitioners to make the best of it)

Closing
- Do you have any questions/concerns?
- End recording.