UMEÅ UNIVERSITY

Developer Experience of a Low-Code Platform: An exploratory study

Daniel Dahlberg

Department of informatics
Master thesis, 30 hp
Master's Programme in Human-Computer Interaction and Social Media
SPM 2020.18

Abstract

In recent years, low-code development has become increasingly popular, enabling developers to write less code and focus on the objective. However, while proven efficient, little attention has been given to how developers experience working in these development environments. This is significant as providing unpleasant experiences could reduce the benefits of low-code platforms by leaving the developers unhappy. As such, this study aims to gain an understanding of developer experience in low-code environments. The study was conducted with an IT-company recently specialized in low-code solutions, where participants were chosen based on having prior experience with low-code development. This ensued in interviews with six low-code developers as well as a project leader. Main positive experiences found were, feeling more productive, improved customer relations, focus on the objective, shared developer understanding, and quick learnability. Key negative experiences found were, having work constrained, limited freedom and creativity, inadequate documentation, and overview, and having poor and unsafe teamwork capabilities. To the best of my knowledge this is the first study to explore developer experience in low-code development environments and stands as groundwork for future studies in low-code developer experience.

Keywords: Business Automation, Developer Experience, Evaluation, HCI, Low-code Platform, System Development, User Experience

1. Introduction

The demand for technological solutions grows larger as the technological landscape reforms how businesses must operate to survive. Software must be delivered quickly, effectively, efficiently, securely and hold quality. Consequently, this has created a higher demand for specialized skills within the tech industry. Businesses must now have IT professions specialized in different development categories, cybersecurity, quality assurance, data, infrastructure and so on. However, as of today, reports suggest that this diverse skill demand has created a deficiency of available workforce within the IT sector (CompTIA, 2019; SHRM, 2019). One potential way of reducing this problem could be to utilize the recent emergence of low-code platforms. Low-code platforms provide companies with the ability to significantly speed up the application delivery, reduce the amount of workforce needed and increase their scalability (Sanchis et al., 2019). This is done by reducing the amount of code needed by breaking down essential functionality into reusable components that quickly can be put together and reshaped (Tisi et al., 2019). Additionally, some low-code platforms bridge and simplifies the gap between areas such as security, data handling and infrastructure, eliminating the need for specialized expertise (Sanchis et al., 2019). However, while studies have shown that these low-code platforms increase the efficiency of application delivery (Waszkowski, 2019), to the extent of my knowledge, little research seems to have been done from a user experience perspective in this area.

User experience has become a very important aspect of software development today (CompTIA, 2019), involving attitudes and emotions of an interaction (Law et al., 2009). Understanding the attributes of these interactions are necessary as they expose ways of how the experience could be improved (Beecham et al., 2008; Kuusinen et al., 2016). Furthermore, research has shown that negative experiences can cause mental health issues among developers, making them take shortcuts in the development process, potentially introducing bad software quality (Graziotin et al., 2017a; Graziotin et al., 2017b).

Now, while user experience is concerned about the contexts in which a system is used, the complex nature of software development involves more than just using a system (Fagerholm & Munch, 2012). Here, developers experience the creation of systems that will be used by others while also iteratively reshaping their own experiences by altering and extending their development environments (Kuusinen et al., 2016). Consequently, in order to fill the gaps of user experience in software development environments, the novel area called *developer experience* has recently arisen, focusing on the activities of software development where traditional UX would not suffice (Fagerholm & Munch, 2012).

Thus, with the importance of understanding how developers experience development, as well as the seemingly limited research, this study aims to understand developer experience in low-code environments. Consequently, based on the dimensions of Fagerholm and Munch (2012)'s developer experience framework, the following three research questions were formed (see table 1).

1.1 Research questions

Nr.	*Research Question*
RQ1	*How do software developers feel about their work in low-code platforms?*
RQ2	*How do developers perceive their contribution's worth in low-code platforms?*
RQ3	*How do developers consider the infrastructure for development in low-code platforms?*

Table 1 – Main research questions

The first question is based on the *affect* category, found in the developer experience framework (Fagerholm & Munch, 2012), and treats factors related to how the developer feels about their work such as sense of belonging, respect or attachment to social connections or the work itself.

The second question addresses the *conation* category and explores how the developers experience their value of contribution. This includes factors such as motivation, goals, alignment, commitment, plan, and intention.

The third and last question looks at the *cognition* category, which is how developers think about the infrastructure of the development process. This category includes factors related to platform, techniques, process, skill, and procedures.

In relation to the three main research questions, two other aspects of developer experience is also of interest as low-code development substantially differ from traditional development (see table 3), moreover, developers may also change their own experiences through individual customization of their own development environment through means such as using and configuring development tools (Kuusinen et al., 2016; Tchounikine, 2017). As a result, in order to understand how experiences may differ between low-code development and traditional development, as well as how developers may reshape their own experiences, the following two research questions have been created (see table 2).

Nr.	*Research Question*
RQ4	*How do developers experience differ between traditional and low-code development?*
RQ5	*How do developers reshape their own experiences?*

Table 2 - Secondary research questions

2. Related Research

In this chapter, definitions and related research of low-code platforms, user experience and developer experience, will be presented to the reader.

2.1 Low-Code Platforms

Development Platforms that reduce code and automate development tasks for developers have recently started to become widely adopted (Koksal, 2020), and is now growing even further due to the outbreak of COVID-19 (Greig, 2020). These low-code platforms can be defined as:

> *"set of tools for programmers and non-programmers. It enables quick generation and delivery of business applications with minimum effort to write in a coding language and requires the least possible effort for the installation and configuration of environments, and training and implementation" (Waszkowski, 2019).*

By simplifying the process of code to goal, developers now need to spend less time figuring out the complex coding requirements of a system and may instead switch their efforts to quickly building new functionality. This is done by automating and standardizing code that effortlessly can be implemented in a visual coding environment.

2.1.1 Differences to traditional development

Low-code platforms are able to significantly change the development landscape in terms of, *time to market*, *scope*, *development*, *maintenance*, *integration,* and *deployment*, compared to traditional development (Morris, 2017). Consequently, it may provide a different experience for the developer compared to developer experience in various traditional development

projects where more traditional tools and processes are used. A summary of the key differences between low-code development and traditional development can be seen in table 3, sourced from Morris (2017).

Feature	Traditional Development	Low-code Development
Time to market	Manual coding Slow to launch Slow to change	Drag and drop design Pre-built components Quick to launch
Scope	Build entire applications at once Large projects	Build small independent solutions one at a time Framework for multiple small solutions
Development	Slow turnaround times Not aligned to business demands	RAD: Build up to 5x quicker Efficient testing Prove ROI
Maintenance	Expensive to support Requires additional development	Easy/quick to update or extend Excellent for prototyping
Integration	Time and investment heavy Requires developer(s) & documentation Open to testing delays	Pre-built connectors Live debugging Create web services/APIs with no coding
Deployment	Slow and complex Multiple steps requiring development resources	One-touch deployment Deploy to multiple environments Cloud or on-premise

Table 3 - How do low-code development platforms compare to the traditional approach? (Morris 2017)

2.2 User Experience

According to the international standard on ergonomics of human-system interaction, user experience can be defined as "person's perceptions and responses resulting from the use and/or anticipated use of a product, system or service" (International Organization for Standardization, 2019). Similarly, Law et al. (2009) describes user experience as a person's interaction with products, systems, services, and objects through a user interface. Consequently, this means that while our full experience may involve experiences such as events, spaces, physical interactions with people, art and so on, they are not part of user experience unless the interaction is done though a manmade user interface. To illustrate, some examples of user experience could be the interaction of playing mobile game, using a self-service checkout machine, or searching for a movie on a streaming platform. During each of these interactions, perceptions and responses are created within the user.

2.3 Developer Experience

The main goal of user experience is the use of a product, system or service. While developers are also users of these products, systems or services, they are also more than a regular user,

they are producers. The developers' interactions with these artifacts are more complex in nature as they both create and use these artifacts in an iteratively changing fashion. Kuusinen et al. (2016) defines it as:

> *"While UX considers the context of use of a system, DX considers the context of software development, including aspects beyond software tools, such as development processes, modeling methods, and other means of structuring SE tasks".*

Additionally, developers also continuously change their own experiences as they often change their developing environment through the configuration and use of tools (Kuusinen et al., 2016). Subsequently, developer experience shares some characteristics with user experience as the focus lies on the user, or the developer in this case. However, the context for developers (software development) is different in comparison to more traditional users, as developers are also producers, requiring extension of our scope in order to understand the factors affecting the experience of software development. Moreover, UX focuses on the use of a product (user-centered design) while DX focuses on the creation of products (process-product-centered in specific contexts). Consequently, UX involve the experiences of using a product whereas DX covers the development processes of creating products, each bringing their own priorities and goals (Fagerholm and Munch, 2012).

2.3.1 Developer Experience Framework

In contrast to user experience, the magnitude of the DX activities not only focuses on the affective attributes, but equally as much on the conative and cognitive as well as social attributes (Fagerholm and Munch, 2012). Consequently, the concept of mind is of importance.

Fagerholm and Munch (2012) breaks down DX into three categories, cognition (How do developers perceive the development infrastructure?), affect (How do developers feel about their work?) and conation (How do developers see the value of their contribution?). These

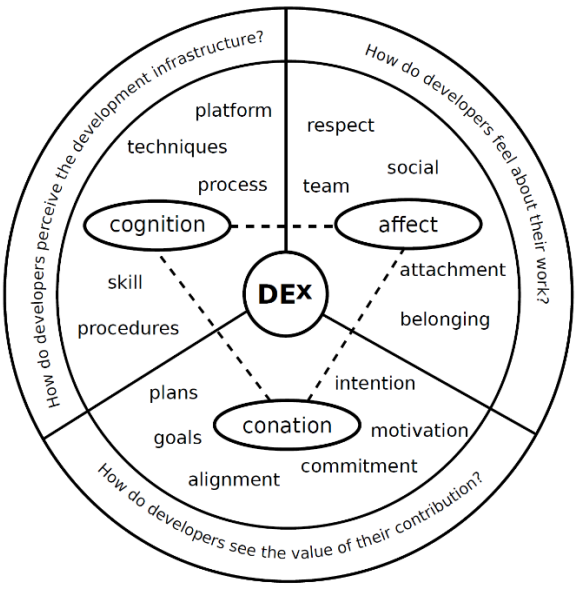

Figure 1 – Developer Experience Framework (Fagerholm & Munch, 2012)

categories are based on the classical separation of the mind, where cognition includes memory, attention, decision-making, problem-solving and the understanding and production of language. Affect, encompassing emotion, feelings and mood. And lastly conation, which consists of motivation, impulse, desire, volition and striving (Fagerholm & Munch, 2012; Fagerholm, 2015).

As seen in figure 1, each of the three sections consists of multiple subsections, exploring how the developer experiences the infrastructure, their work, and their contribution. To further exemplify, the developer may have perceptions towards the infrastructural components such as a platform, programming language, framework, or method, as well as feelings about work, including respect and attachment to their work as well as social aspects contributing to a sense of belonging, and lastly seeing value of their contribution and feeling purposeful by being able to complete personal goals and plans as well as having them being aligned with others. For a more general description see 1.1.

2.3.3 Various contexts

At the time of writing this paper, and to the best of my knowledge, only a limited amount of studies has been conducted with developer experience in mind. Palviainen et al. (2015), investigated how developers experience working in a collaborative online coding environment using Fagerholm and Munch's framework (2012). The results of Palviainen et al. (2015)'s research, further strengthened the statement that developer experience extends further than only experiencing the usage of a tool, it highlights the dynamic nature developers experience, formed by the tools, processes and people as they reiterate their actions.

In a study by Fontão et al. (2017) where they look at the primary emotions in mobile software systems, they focus on the emotions of developer experience. Furthermore, Fontão et al. (2018) employed Fagerholm and Munch's framework (2012) as they explore how developers experience training approaches for mobile software ecosystems, while Kuusinen et al. (2015) conducted a study of developer experience in an integrated development environment in order to improve the tools of the developer as well as their activities. Kuusinen et al. (2015) additionally states the complexity of software development, signifying that developer experience is a very unstudied and critical topic. Continuing, further contexts can be found in Fagerholm's (2015) study, where he explores the developer experience in lean, agile and open source development environments.

2.3.5 Consequences

Investigating the developer experiences is important, as DX may affect the development outcomes in negative ways. Studies have shown that unhappy developers can affect productivity and performance negatively (Graziotin, 2017a; Graziotin, 2017b). Furthermore, it may lead to developers taking shortcuts, impacting the quality of the product (i.e., errors, security and so on) (Graziotin, 2017a). Additionally, it may affect the developer's motivation, resulting in the above issue as well as skipping "unhappy" tasks all the way to quitting their job (Graziotin, 2017a). Furthermore, the tools and their qualities that developers have available may also affect their motivation in positive or negative ways (Kuusinen et al., 2016). Lastly, in a study done by Baltes and Diehl (2018), participants where asked why their own or their co-

workers programming performance may have dropped. The most common answer was demotivation.

Contradictorily, motivated developers are more engaged, focused, and collaborative (França et al., 2014). Furthermore, motivated developers are also better analytical problem solvers (Graziotin et al., 2014).

2.3.6 Motivating factors

As part of exploring the developer experience, it is of relevance to understand what developers in software development are motivated or demotivated by.

By looking at prior research, it is possible to get an overview of the common factors in software development that motivate developers as well as what may cause demotivation. For instance, a study done by Beecham et al. (2008), indicated that the main aspects for motivation among developers were problem-solving, team working, change, challenge, benefit, science, experiment, development practices and software process/lifecycle. In another study conducted by Baltes and Diehl (2018), the following motivating factors were found, presented in order from the most mentioned, problem-solving, seeing result of their work and particularly high-quality work, creating something new, and helping others. In the same study, demotivating factors were also expressed, in order from most mention, non-challenging work, unclear vision where the project is heading, and lack of reward for their quality work. Similarity, França et al. (2014) found similar results, indicating that motivation and happiness comes from, engagement of co-workers, challenging work, social impact, acquisition of useful knowledge, work variety, creativity, well defined work, communication, participation and collaboration, and lastly feedback. Continuing, Misirli et al. (2014) as well as Oliveira and França (2019) found that teamwork was a significant factor for motivation among developers. Other motivational factors found in research showed that developers with more responsibility are more motivated, while processes that improve quality and readiness makes developer less stressed. Furthermore, the study also showed that long production times decreases confidence of the developers (Kärpänoja et al., 2016).

In terms of values toward the technical side, Kuusinen et al. (2016) concluded that developers primarily value *efficiency*, meaning that the development environment should be fast and efficient to use, *flexibility*, covering aspects that fulfil the developer's needs, such as allowing for customization, scalability, being extensive etc., *informativeness*, in relation to code and text editors such as code assistant tools, and finally *intuitiveness*, meaning that the development environment should be easy to use, understandable, intelligent etc.

Technical debt has also been found to be a demotivating factor among developers. *Technical debt* can be defined as the debt given when an easy approach is chosen instead of a more complex and better solution, resulting in more work and costs in the future (Besker et al., 2018). In Besker et al. (2020)'s paper, the results exposed that developers tend to lose morale from technical debt, as it prevents the developer from progressing and reduces confidence. On the other hand, the study also showed that good management of technical debt, can improve the developer's morale by allowing the developers to perform their work better.

2.4 Software and Designing for Appropriation

As developers can reform their own experiences by changing their tools, processes, and environment to their personal liking, it is appropriate to explore how this may take place in a low-code platform. Furthermore, this is also advocated by Fagerholm (2015).

The process of users adopting software in ways that make it their own is defined as *software appropriation* (Dourish, 2003). Note that this is more than a technical change of software, it is concerned with how users adopt based on the context and their needs (Tchounikine, 2017). Consequently, software may be used in ways that the designer did not anticipate for.

While software appropriation is about the personal adoption of software, *design for appropriation* focuses on helping the users adopt a system by intentionally providing means for the user to extend their use in their own unique ways (Tchounikine, 2017). As developers are both users and creators, it is particularly interesting to explore the dimensions in which the developers may shape their development tools as well as the product.

2.5 Developer Experience in Low-Code Platforms

At the time of writing this paper, research on the topic of developer experience seem limited. Additionally, the absence of developer experience research has also been acknowledged by Kuusinen et al. (2015) and Nylund (2020). Furthermore, to the best of my knowledge this is the first study to explore developer experience in low-code development environments.

Consequently, with the increased use of low-code platforms, the seeming lack of studies for developer experience in low-code platforms and the importance of creating good experiences for developers, this study aims to explore and broaden the understanding of DX in a low-code environments through Fagerholm and Munch's framework (2012) (see table 1 for research question one, two and three). Moreover, differences between traditional development and low-code platform experiences will also be investigated as well as how the developers may re-appropriate their tools to shape their experience (see table 2 for research question four and five). Finally, based on these results, recommendations for improving the developer experience in low-code platforms will be made.

3. Methodology

This section exposes the methodological procedure of this study. The chapter is broken down into four main categories, *research design*, containing information about the methodological choices of the study, *research execution*, explaining how the data was collected and analyzed, *sample and participants*, presenting the sampling method and participant selection, *research ethics*, accounting for how ethical research was considered and lastly, *research discussion*, expressing the strengths and weaknesses of the chosen methods.

3.1 Research Design

In order to explore the research questions defined in this study, a qualitative approach has been selected. Qualitative research enables the researcher to explore the perspectives, behaviors, and deeper meanings of people in their real contextual situations (Yin, 2011). As user experience and developer experience revolve around perceptions, feelings, behavior, and

the context of the interaction (Hassenzahl & Tractinsky, 2006), and with the limited resources of this study, it was viable to conduct the research through a qualitative approach based on semi-conducted interviews with developer using a low-code platform called Softadmin. This platform provides developers with standard components that can be used to build technical solutions quickly with high flexibility and scalability (Multisoft, 2020).

3.1.1 Interview Background

This paper uses interviews for understanding the studied phenomena. According to Beck and Manuel (2008), interviews should be used if want to understand humans on a deeper level, when the research question is more interested in the underlying meaning and not the numbers and lastly when investigating trends and themes of experiences. Consequently, as this study is focused on the developer experience and deeper internal thoughts of the developers, it is a practical choice of data collection tool.

3.1.2 Interview Structure

The interviews were conducted with semi-structured questions. By using semi-constructed interviews, the researcher was allowed to create follow up questions and get more detailed answers as semi-constructed interviews are more flexible and deeper than constructed interviews (Beck & Manuel, 2008; Bryman, 2011). The questions became based on Fagerholm and Munch's (2012) developer framework (seen in figure 1), with a distinct focus on the low-code platform environment. Furthermore, the questions were balanced to achieve results in each of the three areas, cognition, conation and affect. More generally, this meant questions related to how the developer consider the development infrastructure (e.g. what is possible to do in Softadmin?), how they feel about their work (e.g. how is it to work in Softadmin?), and how they see value of their contributions (e.g. what have you developed so far?, how did it go?). Many questions would also have overlapping areas and therefore provided richer information.

3.2 Research Execution

The interviews were conducted with a local IT-company located in Umeå, Sweden. For the purpose of improved comprehension and privacy, the IT-company will be named LowCodeTech in this study. LowCodeTech specializes in delivery of IT-solutions, support and hardware and have in the recent years adopted a low-code platform development strategy for delivering technological solutions quicker. As a result of this, a majority of the developers had worked in the platform for a year or longer. Furthermore, the researcher was also working as a part time developer on this firm and already had some experience with low-code development. Moreover, this meant that the researcher was acquainted with the developers.

3.2.1 Data Collection

Each participant was invited to an interview with the information that it would revolve around how they have been experiencing the low-code platform. The interviews were then conducted in private rooms at LowCodeTech. At the beginning of the interview, the participants got handed a consent form, including information about the topic of the interview as well as how their data will be processed (see 3.4 for research ethics). Once signed, voice recording was started on a mobile phone as well as a computer. The researcher would go through each question constructed and ask follow-up questions to get more detailed insights or perspectives.

At the end of the interview, participants were asked for any additional comments and then thanked for their time and input. It should be noted that all information and data was processed in Swedish.

3.2.1 Data Analysis

After each interview, a transcription was made of the recorded data. Once all the interviews and transcriptions had been completed, a coding session would begin. The coding began by highlighting and categorizing comments into their respective area of Fagerholm and Munch's (2012) developer framework (seen in figure 1) for each developer. As this step was finished, the researcher would search for themes in the comments. Each comment then becomes organized into matching themes as well as counted for the total amount of developers expressing that theme. In parallel with searching themes, the researcher analyzed for themes and comments related to design appropriation (see 2.6) as well as differences between traditional and low-code platforms. Furthermore, each comment would in addition to being placed in a theme also get marked as either positive or negative, with the exception of differences to traditional development where also neutral comments could exist. Software and design appropriation would however not hold any positive, negative, or neutral comments as this was not relevant.

3.3 Sample and Participants

This study uses *purposive sampling*. With the limited resources of this study as well as its exploratory nature, purposive sampling is a sensible choice as it allows the researcher, based on personal judgement, to select participants within their operating context (Taherdoost, 2016). As such, participants with a background in low-code development can be chosen to give rich and informational answers practically related to the low-code setting of this study.

The sample selection was made from LowCodeTech that in recent years had adopted a low-code platform. An invitation was sent to all developers who had worked for a year or longer with the low-code platform as well as a project leader who had supervised the work of the low-code platform environment.

A total of seven participants were interviewed. Six of these worked as developers and one as a project leader. This included all developers of LowCodeTech that had been working with low-code development. Five of the developers were men while the project leader and one developer were women. Furthermore, the age range of the participants was between thirty and fifty. Five of the developers were active users of the platform while the sixth had some experience with the platform through integration of external software. All of the developers had between one to two years of experience with the low-code platform and also eight years or more of traditional development experience. Lastly, the project leader had supervised multiple low-code platform projects and was very experienced with both the technical and organizational elements of the platform.

3.4 Research Ethics

In 2002 the Swedish Research Council (2002) published four key principles for conducting ethical research. These principles are presented next.

The information requirement. The researcher must inform the purpose of the research to all those who are affected by the study.

The consent requirement. Participants of a study will always have right to decide over their own participation.

The confidentiality requirement. All information related to people must be given the highest confidentiality possible so that no external part may access them.

The usage requirement. All collected data about individuals may only be used in purpose of the study.

3.4.1 Adoption of guidelines

This previously mentioned ethical guidelines have been considered as this study was design and conducted. The consent requirement was reflected on by informing the participants of the goal and topic of the study, both prior, during and after as participants were handed a copy of the consent form, holding information about the study. Secondly, the consent requirement was fulfilled by having the participants sign a consent form holding information about the study, how the data is handled, their rights, and what they consent to. Thirdly, the confidentiality requirement was respected by storing all the collected data on encrypted devices with two step verification security, ensuring that no external part can access them. Lastly, the usage requirement has been upheld by only utilizing the collected data for the purpose of this study.

3.5 Method discussion

During this study, interviews have been used as data collection method. By selecting this data collection method, a couple of limitations may be introduced as interviews are subject to being affected by bias, resulting in possibly less generalizable results (Bell, 2016). Furthermore, the results could be influenced by the researcher's inexperience of conducting interviews as well as being colleague to the developers, as this could result in data being kept secret (Mcconnell-Henry et al., 2009). Conversely, the researcher did have practical experience of the low-code platform, potentially strengthening the scope and detail of the questions asked during the interviews. Moreover, as the researcher is familiar with the developers in this study, it might have formed trust between the developer and the participant, opening up for a more relaxed discussion, consequently aiding the extraction of more genuine and personal answers (Mcconnell-Henry et al., 2009). Implications for this were suggested by the participants' interest of the topic as well as willingness to attend and share their own inputs. Likewise, during the interviews, participants were eager to share their personal insights and experiences, which was facilitated by the unstrained atmosphere. Lastly, participants could also quickly relate to issues that the researcher also were familiar with, progressing the conversation deeper with few hiccups.

4. Results

This section presents a compilation of the core topics found during the study. This includes the main DX findings based on Fagerholm and Munch's (2012) framework, software and design appropriation and finally traditional development differences. Citations have been translated from Swedish to English.

4.1 Developer Experience

This part presents the main developer experiences found in the interviews. The results are presented in relation to their related research question, based on their placement in the developer experience framework (Fagerholm and Munch's 2012). Furthermore, to create a more comprehensive overview of the results, each finding is also characterized as a negative or positive experience.

The coding of the interviews resulted in a couple of distinct themes that each developer affect could be categorized into and will be described next. The formed themes were the following, *productivity*, which includes experiences about being more productive, *collaboration*, covering experiences about teamwork and communication, *problem-solving*, involving experiences related to solving problems and intellectual challenges, *personal touch*, concerning experiences about being creative and expressing personal touches, *cognitive overhead*, encompassing experiences of having to take more or less steps in order to reach a goal or solution, *common ground*, which covers experiences of sharing the same base knowledge and tools for development, *learning process*, consisting of experiences related to learning and building knowledge, *skill entry*, which holds skill requirements for getting started with the low-code platform, *specialized problem-solving*, concerning low-code elements that require developers to form new specialized thinking processes, *hard dependency*, where issues and possibilities related to control and third party ownership is included, *documentation and overview*, holding topics about documentation and overview of the features, tools and possibilities of the low-code platform, and finally *programming language*, encompassing issues on the subject of programming language.

4.1.1 – Research Question 1

How do software developers feel about their work in low-code platforms?

Theme	Description	Positive developer count	Negative developer count
Productivity	Feelings about being more productive	5	
Collaboration	Feelings about teamwork and communication	2	1
Problem-solving	Feelings about solving problems	3	6
Personal Touch	Feelings about creativity and personal touch	1	5

Table 4 – Feelings about work in low-code platforms

Questions related to how the developer feel about their work in the low-code platform resulted in feelings that could be categorized into four distinct themes, productivity, collaboration, problem-solving and personal touch. Issues related to negative productivity were however not found. The total amount of developers (and the project leader) expressing a theme can be found in table 4.

Positive Experiences:

Four developers as well as the project leader expressed that they felt positive, happy, or satisfied by being able to produce results in a timely, quickly, and easy manner. These results were based on comments such as ”very satisfying to be efficient” (developer 2), “there is happiness in being able to deliver results so fast” (developer 6), “It’s fun to be so productive and being able to skip all the boring parts” (developer 7).

One developer and the project leader mentioned positive feelings about the collaboration around the platform. The results were based on statements such as “they are having fun; it is easy to work together in this and you also get faster and more feedback from the customer which makes one happy” (project leader), “I see it a bit how we manage to fulfill the customer's wishes in a very good way. It kind of brings joy to the development” (developer 6), “it is fun when the customer actually appreciates it… they get a focus on solution rather than having it look a certain way” (developer 6).

Three developers voiced positive feelings about the problem-solving and intellectual challenges that the low-code platform would offer. Some examples are the following, “I felt that I miss it a bit to work with databases. So, then it became like a way in to be able to do that. I think it's fun” (developer 2), “I enjoyed it but it did not go as fast as I did but it was because I was still learning new things” (developer 4), “challenging in such a way that you have to write quite a lot of clever SQL code and it is fun” (developer 6).

The project leader also talked about personal touch, creativity and how developers build an acceptance towards the layout after a while “you get used to a layout… you get used to it looking like this… so that then it will be easier to eat it” (project leader).

Negative Experiences:

One developer mentioned negative feelings about the collaboration around the platform. The results were based on statement, “I'm more into working locally actually” (developer 2).

Five developers and the project leader expressed negative feelings about problem-solving and intellectual challenges, example comments were, “those who like to code raw, they don’t think it is so fun” (project leader), “Sometimes you get a little frustrated when it was not possible to do as you wanted” (developer 2), “I would rather do programming because it's more fun…you get to think more” (developer 4), “You can get really frustrated that you can only get two buttons in the width” (developer 5).

Four developers and the project leader had negative feelings towards creativity and personal touches on the platform. Some key issues where, “changing the interface. Some do not like that you cannot put your personal touch on it” (project leader), “ There is like no creativity.. , it feels like it is very forced how it should be” (developer 3). “a bit boring that that it is left behind… You lose creativity and the artistic part” (developer 5).

4.1.2 – Research Question 2

How do developers perceive their contribution's worth in low-code platforms?

Theme	Description	Positive developer count	Negative developer count
Productivity	Perceived value of being productive	7	
Collaboration	Perceived value of teamwork and communication	4	
Problem-solving	Perceived value of solving problems	4	6
Personal Touch	Perceived value of creativity and personal touch		6

Table 5 - Perceived value of work in low-code platforms

Questions related to how the developer perceive the value of their contributions resulted in four distinct categories, productivity, collaboration, problem-solving and personal touch. Perceptions related to positive personal touch were however not found. The total amount of developers (and the project leader) expressing a theme can be found in table 5.

Positive Experiences:

Six developers and the project leader reported positive perceived value of their work by being more productive. Some examples were: "You can do a lot in one day. In one day you are able to do very, very much" (developer 2), "very easy to do things…fast" (developer 3), "you start on a more helpful percentage…it is quick to prototype for the customer" (developer 6), "productive that you can do a lot with little time…and also flexible…easy to make changes afterwards" (developer 7).

Three developers and the project leader stated that they had gained positive perceived value of their work by having better teamwork and communication. The results are based on statements such as, "you get very easy and good feedback…so that it turns into what they want" (project leader), "you can sit and look it up and visualize for the customer what your thoughts are" (developer 2), "you can get customers to agree that it should look exactly like this" (developer 3), "You have to say no to customers…that is not a good idea. Significantly earlier than you would otherwise. On the other hand, there is a strength in being able to say no that the platform does not support this, it is not possible" (developer 5).

Four developers expressed positive perceived value of their work by problem-solving and intellectual challenges that the platform provided. Some of the comments were, "you can instead see the advantage of being able to help an organization with very, very simple means" (developer 5), "it is surprisingly much in these requirements that customers have that can actually be solved with what you have to work with. (developer 6), and lastly:

> *"my experience with some other platforms like this is that if you do not do exactly what the people who made the platform thought you should use that file for then you have to work against it rather than with it to make it work the way you want. I think this happens much less often with Softadmin than with others I have tried"* (developer 7).

Negative Experiences:
Five developers and the project leader mentioned that they have had negative perceived value of their work as a result of problem-solving and intellectual challenges. The following examples occurred, “Sometimes it feels like you have to make pretty ugly interfaces, I mean not visually ugly, but they get a little bit harder to work with” (developer 2), “But you cannot…it's pretty square” (developer 3), “That's the thing…if it does not work in Softadmin…then it does not work in Softadmin…there is nothing you can do” (developer 6).

Five developers and the project leader reported negative perceived value of their work because of issues with creativity and personal touch. Some of the key issues were, “you are basically not allowed to design anything” (developer 5), “you do not have much freedom when it comes to what it should look like” (developer 6), “what the interface should look like…it is complicated to fix in Softadmin. So that is a little worse” (developer 7).

4.1.3 – Research Question 3

How do developers consider the infrastructure for development in low-code platforms?

Theme	Description	Positive developer count	Negative developer count
Cognitive Overhead	Attitudes on infrastructure & total steps to reach a solution	5	4
Documentation and overview	Attitudes on infrastructure, documentation & overview		4
Common Ground	Attitudes on infrastructure & shared base knowledge	5	
Learning Process	Attitudes on infrastructure & learning process	5	2
Skill Entry	Attitudes on infrastructure & required entry skills	3	
Programming Language	Attitudes on infrastructure & programming language		3
Specialized Problem-solving	Attitudes on infrastructure & specialized thinking processes	2	
Hard Dependency	Attitudes on infrastructure & third-party control	1	3
Collaboration	Attitudes on infrastructure, teamwork & communication	3	5

Table 6 – Attitudes on infrastructure in low-code platforms

Questions related to how the developer consider the infrastructure for development in low-code platforms resulted in a total of nine categories, cognitive overhead, common ground, learning process, skill entry, specialized problem-solving, hard dependency, collaboration, documentation and overview and lastly programming language. The total amount of developers (and the project leader) expressing a theme can be found in table 6.

Positive Experiences:

Four of the developers and the project leader reported positive attitudes towards infrastructure and total steps to reach a solution. Some key statements were, “But you do not have to put a lot of work into the basic…because it already exists” (project leader), “you write code in SQL, you write questions to be done against the database so you just have to focus on that” (developer 4), “you do not have to think about the interface composition” (developer 6), “Less risk to do errors” (developer 7).

Four of the developers and the project leader told that they have had positive attitudes towards infrastructure and the shared base knowledge it brings. The results are based on some of the following comments, “It is good that everyone involved has the same basis” (project leader), “if we make a new system then there will only be very low entry time for those who worked in the old as well as you get the uniform look of everything” (developer 2), “It is very, very easy for me to hand over a Softadmin solution to another developer who has just completed the training” (developer 5).

Five developers expressed positive attitudes towards infrastructure and the learning process. The result are based on some of the following remarks, “So it was very good because you could basically read the whole thing and work yourself without much help” (developer 2), “everyone takes a course for a few days and then you can use it” (developer 3), “Softadmin is built on standard components. It is very easy to adapt even if you do not have the knowledge” (developer 5), “It was good. It was great educational resources” (developer 6).

Three developers mentioned positive attitudes towards infrastructure and skill requirements. Some key examples are the following, “It is required that you know SQL in order to be able to develop in it, but it is also the only thing required in general” (developer 2), “Anyone really if you are a little learned in programming…if you work with SQL questions you can learn it fast I think. It should go fast” (developer 4).

Two of the developers stated that they positive attitudes towards infrastructure and its specialized thinking processes. Some of their key remarks were, “it is a little different thinking process… I think it forces one into a good mind in some way. The first thing you do is sit down and map out the system” (developer 2), “you think like how do we solve this based on what we have… you need to have a Softadmin mindset in your head when you think about how to solve it” (developer 6).

One developer also showed positive attitudes towards infrastructure and dependency on third-party platform owners. This is based on the following statement, “So they update, they maintain the product and you do not need us to do it more than clicking the update button on the customer and it is fixed as well” (developer 2).

Three developers expressed positive attitudes towards the infrastructure’s teamwork and communication possibilities. Some key examples were, “So you sit at customer meetings. Someone can sit and fix the thing that the customer is just talking about and as well show on the screen directly” (developer 2), “in five minutes from changing it in the data model to adding it to the form and you can view it in the app” (developer 5), “when customers see okay…they may not like exactly what everything looks like but when they see the benefits and that it is still a very easy-to-use interface, then it does not matter much” (developer 7).

Negative Experiences:

Three developers and the project leader stated negative attitudes towards infrastructure and its documentation and overview. The following are some key examples, “it still requires that you work with it for a while to understand” (project leader), “everything is in a database…in one place…it is like you can structure things with schemes and so on, but it's hard to get any bigger order in it” (developer 2), “sometimes they have had lacking documentation and you had to email them” (developer 6).

Four developers mentioned that they have had negative attitudes towards infrastructure and total steps to reach a solution. Some statements are the following, “if you then in the middle of the project realize that you have misunderstood something big…then it will be quite messy to correct it later” (developer 2), “It is difficult with this to be able to see what changes I made here to destroy this, how can I diff it and who has done this, how it has happened and so on. Such things are difficult” (developer 5), “Multisoft's app…like tables were limited to 3 columns…(to solve it)…either build your own mobile app as we have done or do your own APIs that do things” (developer 7).

Two developers expressed negative attitudes towards infrastructure and the learning process. Some of the comments where, “I feel it may be a small disadvantage that the introductory material goes through everything quite clearly, but where It ends, there you are rarely working yourself” (developer 2), “I think learning SQL and SQL server and how to do things there in simple and efficient ways…that's what is difficult” (developer 7).

Three developers said that they have had negative attitudes towards infrastructure and its programming language. The results are based on some of the following, “SQL is the basis of SoftAdmin and it is a huge big thing…I think that few can call themselves fully educated in SQL as well” (developer 2), “SQL is in many ways superb but in many ways useless, most primitive programming language and using SQL as a programming language means that you inherit many of the qualities of the negative aspects that SQL has” (developer 5).

Two developers and the project leader stated negative attitudes towards infrastructure and the dependency on third-party platform owners. This is based on some of the main comments, “Yes, then a potentially bad thing is that the whole tool is one…yes, it is like a third party tool the whole platform…you sit a little in the knees of them” (developer 2), “But I believe that you could open it up and make it less dependent and take it to another level with the possibility when it comes to customizations and interfaces and stuff” (developer 5).

Five developers expressed negative attitudes towards the infrastructure’s teamwork and communication possibilities. The following key statements occurred, “it's a bit dangerous to be work in such things…if you save a stored procedure that I also open and maybe I am unsure if I have saved this or not…then an hour later I save it and overwrite your changes” (developer 2), “I make a change and then someone decides that now we will deploy, then my small changes will be included…even though they are not really ready” (developer 4), “to be more than two, three people in a Softadmin system at the same time and actively develop it… then you have to document it very well and have a data-model in place from the beginning” (developer 5).

4.2 Traditional Development and Low-code Development

4.2.1 Research Question 4

How do developers experience differ between traditional and low-code development?

The participants also stated differences in their experience between traditional and low-code development. A total of seven distinct themes could be found from these results, productivity, common ground, creativity & problem-solving, learning process, control, collaboration, and finally "other platforms". Descriptions for these themes can be found in 4.1 with the exception of creativity & problem-solving which includes both problem-solving as well as personal touches (both themes found in 4.1), control meaning changed control and overview and finally other platforms for differences to other platforms. Each theme may be presented as either positive, negative, or neutral difference.

Six developers and the project leader expressed positive differences to traditional development through productivity. Some major statements were the following, "there is nothing that is so easy to start up in" (project leader), "You can do a lot in one day…compared to traditional development where you have to sit with API calls and validations and interfaces that must correspond to both validation, calls and structure" (developer 2), "it seems to go very fast to get something to show…in comparison to when you have to code it yourself from the beginning" (developer 3), "this agility and the speed of meeting changes in customer requirements" (developer 6).

One developer talked about positive differences by common ground. The result is based on the following comment,

> *"if you have made more than one softadmin system to a customer then they will of course recognize themselves in it…then we make a new system then there will only be a very low run-in time for those who worked in the old as well as you get the uniform look of everything. So it's a positive one. It can be done in traditional coding as well, but they become a little more automatic with softadmin" (developer 2)*

Three developers mentioned positive differences in creativity and problem-solving. This is based on some of the following key observations, "you can do some things outside the softadmin sandbox. Get some free hands" (developer 2),

> *"you think like, how do we solve this based on what we have…which makes you get a slightly different focus than that…with traditional development where you maybe do something because you can but then maybe it was not really the best way but you do it anyway because it works"* (developer 6),

"for the type of system where Softadmin fits well, it is difficult to find something that is much better with traditional development" (developer 7).

Five developers did also express negative differences in creativity and problem-solving. These are some of the main comments, "Because you may have encountered a certain situation that could be solved quite easily but then it turns out that there was really no support for that" (developer 2), "you are basically not allowed to design anything" (developer 5), "with

traditional tools…there is maybe a lot more information to access and how others solved things and stuff" (developer 7).

Three developers had additionally neutral differences in creativity and problem-solving. This are some examples, "So I see nothing that traditional coding cannot do that Softadmin can do. After all, there are no restrictions in the traditional world. But it takes much longer" (developer 2),

> *"in traditional development you are paralyzed by having all the possibilities, and you are paralyzed by the paradox that you do not know how best to do something, so instead of doing something fast you try to find the best way to do it, so it takes a very long time… and the other limits one so that one can only do it in one way" (developer 5).*

One developer mentioned positive differences with the learning process. This is based on the following comment, "easier to learn it than to start programming in standard traditional tools" (developer 4).

One developer showed positive differences regarding control. The statement made was,

> *"Because there is so much more code based on many different components that you have downloaded from different manufacturers, so you have to keep track of yourself that they are safe and work over time, while Softadmin releases new features regularly and very simple process to upgrade. It did not get much better for traditional development…for this type of system" (developer 7).*

Three developers talked about positive differences in collaboration. These are some of the main observations made, "And just to avoid this extra labor…I have to send an email to the customer later and describe with arrows and screenshots that we have changed this and does it look good? And instead of being able to do it in a sitting meeting, it is invaluable" (developer 5),

"*I can well think that the need to prototype is not there in the same way. But with Softadmin solutions…if you have shown how Softadmin works and looks…then the customer knows when you explain…then you don't need to mockup layouts and stuff*" (developer 6).

Four developers did however mention negative differences in collaboration. The results are based on some of the following comments, "Well, there is no version control…it must be seen as a big disadvantage that you can never sort of check what others have done or go back" (developer 3),

> *"In traditional, there are millions more functionality and support and stuff for collaboration. Everything from pair programming to that you can sit in different geographical locations and code in the same code while it runs live and you can get branches, you can get an all-DevOps spirit that has ever existed. Not to mention tests and all that stuff. I mean it is thirty years back…forty years back for Softadmin on that part" (developer 5).*

Furthermore, all four developers mention negative differences with version control.

One developer had a neutral stance differences in collaboration. This is based on the following comment:

> *"The easiest way to work with collaboration is to sit close to each other, talk constantly and work with separate parts of the same system, but to be more than two, three people in a Softadmin system at the same time and actively develop it... then you have to document it very well and have a data-model in place from the beginning" (developer 5).*

The project leader mentioned negative differences to another more traditional development platform. The project leader said the following, "SharePoint is a variant, but it is much...much slower and much harder and much more boring to work in as a developer" (project leader).

4.3 Software and Design Appropriation

4.3.1 Research Question 5

How do developers reshape their own experiences?

The participants did express cases of software and design appropriation. These results can be categorized into three main themes, creativity extension, which includes new thought processes that the developer must use in order to solve some problems in the low-code platform, environment extension, which encompasses how the developer extend and reshape their development environment in order to solve problems, and finally productivity extension, holding tools and software that the developer may use to speed up their development.

Two developers made comments about how the low-code platform may require a creativity extension. The comments made are the following, "I know that there are people who have done work-arounds to get around it and it is good that it is possible, but for me it feels like it might be a bit too much hand-laying call to get to something that should be quite easy" (developer 2),

> *"then you try to model the system based on your vision in ordinary cases when it comes to traditional programming. But when you work with Softadmin, it is rather, okay...I have these puzzle pieces and what can I do with these puzzle pieces? Ah, then it must be like this. So that it is a state of mind where you change and it becomes...but and the other side, there are many who say that with creativity arises from limitations" (developer 5).*

Three developers talked about how they had extended their development environment in the low-code platform. This is based on the following statements, "you can do some things outside the softadmin sandbox or what should I say...get some free hands" (developer 2), "precisely because of what I said that SQL is not really built to write algorithms and because of that it will be easier and simpler to test, more separated and nicer if you write certain code in C# instead" (developer 5), "well what we have done so far is to do it outside Softadmin in such cases. With...well, either build your own mobile app as we have done or do your own APIs that do things" (developer 7).

Four developers made comments about tools and software they use to further improve the productivity of the low-code platform. Some key examples are the following, “but we run this..what is this called...SQL Prompt to get intellisense and be able to write a little faster” (developer 2), “we have bought a single component called SQL promt that handles how you code and gives you tips” (developer 5).

5. Discussion

This chapter analyses and discusses the findings of this study based on related research and empirical conclusions.

5.1 Developer Experience

This section holds a discussion on the first three research questions, relating them to research within the field.

5.1.1 Research Question 1

How do software developers feel about their work in low-code platforms?

From the results it is possible to extract that a majority of the developers feel satisfaction or happiness from being more productive in the low-code platform. Which is in line with Fontão et al. (2017)’s result where developers showed joy from improving their productivity, as well the results by Kuusinen et al. (2016), indicating that developers get motivated by an efficient development environment. Additionally, some positive emotions also existed for the increased collaboration abilities with the customer as well as problem-solving and intellectual challenges from working with the programming language of the platform. Subsequently, this is consistent with developers being highly motivated by problem-solving and teamwork (França et al., 2014; Baltes & Diehl, 2018; Oliveira & França, 2019) As for creativity, this is a more neutral statement by the project leader, suggesting that some acceptance towards less creativity grows as productivity replaces it. This is an interesting aspect in its own.

In contrast, negative emotions were significantly expressed for problem-solving and personal touch, as a majority of the developers felt that the platform sometimes constrained their abilities to come up with solutions, along with the fact that they could not use their creativity and personal touch to alter the layout of the product. Successively, this a similar result was shown for emotions in mobile software ecosystems by Fontão et al. (2017), where two of the most common negative emotions, sadness and anger, were related to issues in manipulation of interfaces and incompatibility with the developers’ preferred problem-solving methods. Additionally, the most common positive emotion, joy, was in addition to being more productive, also connected to creativity and being able to manipulate interfaces. Furthermore, this result may also be explained by the fact that creativity is an important factor for motivation (Beecham et al., 2008; França et al., 2014; Baltes & Diehl, 2018). As well as giving the developers freedom (Kärpänoja et al., 2016). Only one developer expressed negative emotions about collaboration, preferring to work locally.

5.1.2 Research Question 2

How do developers perceive their contribution's worth in low-code platforms?

In terms of how developers see the value of their work, all of the developers saw increased productivity of their work. This was also endorsed by the project leader. This may be a major positive impact for the low-code platform as Kuusinen et al. (2016) showed that developers value efficiency in their development environment, while Baltes and Diehl (2018) found that developers are highly motivated by seeing result of their work. Furthermore, Kärpänoja et al. (2016) found that longer production time makes developers less confident. As for collaboration, multiple developers felt that the relationship with the customer had improved by the added transparency, communication, and feedback. This made it possible for the developers to better understand the requirements, reiterate on feedback, hence improving their work efforts. Going back to motivation, developers who experience good communication, feedback, teamwork and have well defined work are according to multiple studies more motivated (França et al., 2014; Baltes & Diehl, 2018; Misirli et al., 2014). Consequently, the low-code platform may provide some motivation through these experiences. Furthermore, some developers also stated that their work had improved as a result of the efficient problem-solving possibilities of the platform. Because of this, developers could now solve problems, see result of their work, and provide the customer with a solution more quickly. Hence, as seeing result of ones work as well as helping others are important factors for motivation among developers (Baltes & Diehl, 2018; Fontão et al., 2018), this provides another strength in terms of developer experience. Additionally, this may improve the moral among the developers as it could reduce technical debt (Besker et al., 2020).

Looking at negative perceived value of the developer's work, almost all developers expressed that they have had their work constrained as a result of the limited choices for problem-solving on the platform. Subsequently, almost all developers also expressed negative attitudes towards the restricted creativity and space for personal touch. As mentioned earlier, problem-solving and creativity is an important aspect of motivation for developers (Beecham et al., 2008; Baltes and Diehl, 2018). Consequently, these are two significant factors that may demotivate the developers in a low-code platform.

5.1.3 Research Question 3

How do developers consider the infrastructure for development in low-code platforms?

For attitudes towards the development infrastructure, the developers had numerous inputs. To start with, a majority of the developers felt that the low-code platform had improved their cognitive overhead by removing and automating steps in the development process, meaning that developers now were able to focus more on the task at hand. Subsequently, as Kuusinen et al. (2016) showed, efficiency and intuitiveness of the development platform is highly valued by developers, suggesting this a major positive experience towards the development infrastructure. Furthermore, as less steps needs to be taken for a solution, it could reduce the technical debt as there is less options for creating the technical debt (Besker et al., 2020). Continuing, the results also indicated that having a common ground was a beneficial key attribute of the platform as the shared base knowledge and functionality allowed the developers to jump in, out and between projects without or very little effort. Furthermore, in relation to this, many developers thought that the learning process was quick, easy and

straightforward while, some developers also talked about the low-skill entry for learning how to develop on the platform, as well as gaining a specialized thinking process. This seem to be the result of the fact that the platform provides limited but easy and efficient tools for the developers to learn and employ. Because of the limited thinking space, the developers are now challenged to think with what they have, making them all share the specialized problem-solving techniques offered by the platform. Moreover, the collaborative features compare to the result of Palviainen et al. (2015), where developers in a collaborative coding environment saw increased efficiency of coordination tasks as well as being more motivated by perception of working together with others. This illustrates that the limited but efficient platform capabilities may allow for more close teamwork and thus improve on the developer's experience. Furthermore, some developers also appreciate the collaborative ability to quickly prototype and work closely with the customer by showing them progress, ideas, and goals. Subsequently, these features may help to further motivate the developers as teamwork, having a clear path, and seeing results of work are important motivators (França et al., 2014; Baltes & Diehl, 2018). Lastly, one developer was also positive to the third-party ownership of the platform, as this meant that the platform would be kept updated by the owner, removing this work from the developers. This could also mean less technical debt as there it is managed by the platform owner, thus improving the morale and work efficiency of the developers (Besker et al., 2020).

Looking at the results for negative attitudes towards the development infrastructure, it is possible to see that there were multiple areas of concern. Firstly, multiple developers as well as the project leader stated that it can be hard to get an overview of the assets used on the platform as well as finding information about them, as the documentation was limited. Fundamentally, this can be seen as a major drawback of the infrastructure and developer experience as Fontão et al. (2017) showed that anger among developers was closely related to unavailability of documentation and not knowing how to proceed. Secondly, several developers also pointed out that the platform's infrastructure sometimes limits their problem-solving, thus increasing the cognitive overhead. While the constrained nature of the platform seem to have created positive effects, as discussed previously, it seems that it also introduces problems and extra steps for the developers to solve, in other words, it may create some technical debt. This is also hinted by the following developer comment, "*if you then in the middle of the project realize that you have misunderstood something big…then it will be quite messy to correct it later*" (developer 2), because options for resolving the issues may be limited in comparison to traditional ways. Subsequently, as developer anger also is associated with incompatibility of resources (Fontão et al., 2017), the fact that developers highly value flexibility of their development environment (Kuusinen et al., 2016), and that technical debt is bad for developer morale (Besker et al., 2020), this suggests that this is another important factor of infrastructure affecting the developer experience. Continuing, while the limited, easy, and efficient scope of the platform had benefits in aspects such as quick learning and skill entry, it can also be seen that developers dislike some of this limitations. For instance, some developers felt that the learning process was quick but did not cover more advanced aspects that later would be become a challenge, as documentation was hard to find. Furthermore, in some relation to this, the programming language of the platform, SQL, was mentioned

numerous times as a constraining factor as it is a very old and advanced programming language. According to the developers, this meant that the developers have to work with the outdated and bad aspects that SQL has. Furthermore, as SQL is a very advanced language, the learning process cannot cover many of the more complicated but important techniques, thus preventing the developers from reaching optimal performance. In relation to Kuusinen et al. (2016)'s research, it could be said that this is a limitation in the flexibility and intuitiveness of the development platform, as well as a risk for technical debt from non-optimal solutions in SQL. Consequently, this is further a key aspects of infrastructure that may affect the developer experience negatively, as a bad learning process may lead to technical debt ending in morale loss (Besker et al., 2020), while the programming language itself as well as having inadequate skills in it, may result in unalignment between the values of the developer and the development environment's offerings. Next, there also seem to be some negative attitudes towards the third-party dependency as this type of infrastructure restricts the developers from having the flexibility they would like when developing a solution. As previously discussed, flexibility (Kuusinen et al., 2016) and freedom (Kärpänoja et al., 2016) in the development environment is something that developer value greatly. As such, these result further suggest these values. Lastly, there were a significant amount comments about a particular collaboration issue on the platform. This issue was related to the fact the platform had a very lacking support for safely working together. Thus, risks such as overwriting someone else's work existed as well as poor overview of what others are doing, have done, or will be doing. Subsequently, in relation to previous research and numerous comments of it, this rises multiple flags for possible demotivation. For instance, one of the four important factors that developers' value is informativeness in their development environment (Kuusinen et al., 2016). The results found here suggest a lack of informativeness for organized collaboration. Furthermore, this issue may prevent good teamwork, an important motivator for developers (Misirli et al., 2014; Oliveira & França, 2019), as well as conceivably causing demotivating by removing the reward for their quality work (Baltes & Diehl, 2018), since their efforts may get overwritten. Consequently, with the numerous comments about this issue as well as the negative effects mentioned earlier, this signifies a major infrastructural component that affect the developer experience negatively. Furthermore, this could also explain why one developer preferred to work alone.

5.2 Traditional versus Low-Code

This section discusses differences found between experiences in traditional and low-code development.

5.2.1 Research Question 4

How do developers experience differ between traditional and low-code development?

Based on the result, it was possible to extract multiple differences that the developers had experienced. Firstly, a key difference found was in productivity, as a majority of the developers felt that they were able start, change and finish significantly more quickly. Secondly, as discussed in 5.1, common ground was found to be a positive factor by many. However one developer stated a direct comparison, expressing that the low-code platform is able to create a common ground between customer and the developers, thus making it easier to for the customer and developers to know the possibilities and constraints of future solutions. Thirdly,

developers found that the limited creativity and problem-solving capabilities of the platform allowed them to focus on doing solutions in the best way possible whereas traditional development would have exposed them to a multitude of options, making it harder to know which approach would be the best. Furthermore, some possibilities did exist for extending the low-code platform with custom code, which to some extent meant that developers could combine some traditional coding with the low-code platform. This is an interesting aspect as it could be seen as a hybrid way development, further rising an issue of balance between creativity and productivity. On the same topic, many developers also had negative comparisons as the limitations of the platform meant that some things could not be solved or designed in ways that the developer originally had in mind. Moreover, traditional development would offer a lot more options for finding information and help for progressing a solution. Continuing, some noteworthy neutral differences could also be found on this topic, containing the issue of infinite possibilities versus development time. As previously discussed, this could be seen as another view of the balance between creativity and productivity. On the next topic, learning process, the discussion in 5.1 argued that the learning process for the low-code platform was very fast. It is possible that the learning process may be faster in this low-code platform as opposed to traditional development considering that this was specifically stated by one developer. In relation to this, it is important to note that all of the developers had many years of experience with traditional development. Going further, the difference of control between traditional and this low-code platform is another changed aspect of the overall development experience as there is less things that developers have to consider in terms of security and updates. According to one developer this was a positive difference to traditional development as it meant less work as well as future work, such as technical debt (Besker et al., 2020). Another topic that the developers found different was collaboration, including positive negative and neutral aspects. From the positive comments it is possible to see that the easy prototyping, available on low-code platforms (Morris, 2017), and common ground of understanding had improved the communication with the customer by being able to do direct feedback changes as well as the customer understanding the possibilities and limitations of the platform. However, a major negative difference stated by many developers was the fact that there was a lack of support for safe and sound teamwork on platform, compared to traditional development where a magnitude of collaboration tools would be at hand. Consequently, this may form concerns among the developers as traditionally code can be tracked, reverted and worked on simultaneously while their low-code development efforts could get wiped by another developer not knowing that that the work already had been completed. Furthermore, it also means that it can be hard to track what went wrong, when did it go wrong, and who did it. Nevertheless, a developer did state in a neutral comment 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. Lastly, the project leader mentioned that the team previously had been working with another development platform, holding some similarities to a low-code platform. Evidently, Softadmin was a lot easier and more fun to work with.

5.3 Software and Design for Appropriation

This section discusses findings of software appropriation and design for appropriation in the low-code platform.

5.3.1 Research Question 5

How do developers reshape their own experiences?

The findings in this study shows that the developers themselves employ software appropriation (Dourish, 2003) to reshape their experiences. Furthermore, the low-code platform seems to include some design for appropriation (Tchounikine, 2017), enabling developers to reshape their development environment and processes. The results could be categorized into three areas, the extension of creativity, environment, and productivity.

The limitations set by the platform appear to have created a form of creativity where developers have to find their own ways around problems by bypassing them through creative means, sometimes not intended by the low-code platform. Consequently, these creative solutions may sometimes require the developer to extend their development environment by developing outside and in parallel with the low-code platform. By allowing the developers to partly extend the platform's functionality, the low-code platform have utilized some design for appropriation as each developer are able to slightly reshape and extend their development environment to their own wishes. Lastly, while the low-code platform has created a feeling of increased productivity, the developers have found further ways to extend the productivity by adding external tools that help them code faster and smarter. However, this is a design for appropriation possibility given by Microsoft SQL Management Studio (a tool for database management) and not the low-code platform itself. Summarizing, it is possible to see that the low-code platform provides some design for appropriation and that developers exploit this for their software appropriation in order to support their individual creative goals as well as being more productive.

5.4 Consequences

Providing developers with a good experience is important as it may reduce the negative effects from bad experiences as well as enhance the positive effects of having good experiences (see 2.3.5 where this issue is described). Consequently, it should be reflected on how the results of this study may impact the developers. Therefore, a discussion will be held here on the possible consequences that may occur as a result of the perceived developer experience in the low-code platform.

5.4.1 Key factors for positive outcomes

As shown by França et al. (2014) and Graziotin et al. (2014), motivated developers are more engaged, focused, collaborative, and better problem-solvers. The results of this study indicate that there may me several implications for motivation and positive emptions when working with the low-code platform. As discussed in previous sections it can be concluded that some of the key positive values were, being more productive, improved relationship with the customer, having to focus less on boring and redundant steps, having a common ground of knowledge and being able to learn it quickly. Consequently, as it has been discussed, these characteristics may enable the developers to gain positive experiences and motivation, indicating that they are

the key strengths for positive experience on the platform and may contribute to improved performance among the developers.

5.4.2 Key factors for negative outcomes

Unhappy and demotivated developers may reduce the developer's productivity and performance (Graziotin, 2017a; Graziotin, 2017b; Baltes and Diehl, 2018) as well as reducing the quality of the product (Graziotin, 2017a). While the results showed multiple positive consequences for experience, it also held a few negative aspects that could create demotivation and negative feelings. The key findings were, having their goals constrained, restricted creativity and personal touch, poor documentation, and overview as well as poor and unsafe teamwork. As discussed in previous sections, these may evidently cause demotivation among developers contributing to the negative developer performance.

5.5 Recommendations for low-code platform owners

Based on the key strengths and weakness found in this study, a couple of recommendations can be made for low-code platform owners, wishing to improve on the developer experience.

A key challenge that should be considered is the balancing act between productivity and freedom on the platform. As it has been discussed, while the limitations enable the developers to create optimal and speedy solutions, it also may constrain productivity by hindering the developers from achieving a particular goal, hence making the developer do workarounds and possibly suboptimal solutions in order to continue. Subsequently, it also prevents the developer from expressing their creativity as well as putting their personal touch on the product, which may demotivate the developer (Beecham et al., 2008; Baltes and Diehl, 2018). On the other hand, giving too much freedom and creativity could diminish the productivity gain as developers now have too many options, making it more difficult to select the best approach, thus making the project timeline longer and more complex, which may demotivate the developer (Kärpänoja et al., 2016; Besker et al., 2020). It may however boost productivity in some cases as developers have an easier time reaching their goals. Furthermore, allowing more freedom and creativity could make the platform lose some of the effects from common ground as developers lose some of the shared knowledge, enabling them to jumping in or out between projects, as each developer now can create their own personal solutions and looks. Still, more freedom and creativity may allow the developer to become more motivated (Beecham et al., 2008; Baltes and Diehl, 2018) as it is easier to put their personal touch on the product. This is a twofold issue where the platform owners must balance limitation against freedom in order to ensure that development is efficient and productive but still allows the developers to express their creativity.

Teamwork is an important aspect for motivation developers (Misirli et al., 2014; Oliveira & França, 2019). This study showed that developers of the low-code platform were happy with the improved customer relationship as they could prototype more easily. However, teamwork fell short when it came to developers working together as the collaborative environment was unsafe and unclear to work in. As such, platform owners should focus on making good collaborative features for developers to feel the pretense of others, having a safe development environment for teamwork and making the platform informative of what other are doing, have done or will be doing.

The platform owner should ensure that proper documentation and support is in place as it otherwise prevents the developer from progressing and causes negative feelings (Fontão et al., 2017). Furthermore, it may help the common ground effect because there will be less workarounds done by each individual developer. Additionally, it should be easy for the developers to get an overview of the components used in the platform as informative development environments is valued by developers (Kuusinen et al., 2016).

Providing developers with an easy but still rich learning process could help to improve the common ground among developers as every developer can proceed where another developer left off. Moreover, it decreases the need to read up on documentation for the developers in later stages, where documentation, as mention before, must be of proper quality.

6. Limitations and Future Work

This section discusses the potential limitations bound to this study as well as areas for future research opportunities. For methodological issues (see 3.5).

6.1 Limitations and gaps

With the so far limited research on the topic of developer experience, it can be difficult to compare and make draw generalizable conclusions for developer experience, and in particularly for low-code environments. Furthermore, this have made it difficult to estimate how significant the lesser mentioned experiences may be, as further studies are needed to confirm the weight and scope of these issues. However, the exploration done by this study has opened up the landscape for studying developer experience in low-code environments and may stand as a starting ground for future researchers to investigate and compare developer experience in low-code platforms as well as other developer environments.

This study was conducted in collaboration with a smaller IT-firm where the researcher also worked. As this may have an impact on the result, future researchers should be aware that results could be different in larger and more complex firms. This provides an opportunity for future research on the differences. Furthermore, all developers of this study had multiple years of experience in traditional development and a few in low-code development. This may further have an impact on the results and should be further investigated by exploring how prior experience, age, gender, or possibly development preference may impact the developer experience.

Lastly, the research methods used in this study have shown to be effective in exploring the general developer experience in the low-code environments. However, as discussed, user experience is complex, and even more so for developers, and thus these methods may not encompass the whole experience. Researchers interested in developer experience therefore have an opening for considering different methodological tools in order to fill the gaps of this study.

6.2 Topics of interest

In addition to the research opportunities mention above, a few topics of interest have occurred during this study. Firstly, the project leader in this study had an interesting comment about

how developers may drop creativity for productivity. This raises the question if some elements of experience can lessen the need for other developer experiences. Future researchers may explore how developer experience weight and compare against each other in different developer environments. Furthermore, this study showed that there is a balancing act between productivity and creativity in low-code environments. An interesting issue is, what is enough creativity for developers? Future research could help low-code platform owners by investigating how much creativity should offered in order to maintain motivation among developers.

7. Conclusion

This section concludes the paper by briefly describing the study and its outcomes.

This paper has been concerned with how developer experience takes fold in low-code environments. Using Fagerholm and Munch, (2012)'s framework as foundation, the study explored developer experience among six developers and a project leader that had been working with a low-code platform in the recent years.

The findings indicate that the major components for positive developer experience are being able to be more productive, having an improved relationship with the customer as prototyping and showcasing becomes easier, being able to focus on the task at hand and put less effort into previously required tasks as these have been automated, making it easier start and switch between projects, as skill and knowledge is shared among developers, and finally being able to learn it quickly.

Negative factors for developer experience were also found. The main findings include, having their work and goals constrained by limitations set on the platform, having less freedom and creativity for personal touch as a result of the restrictions set by the platform, unable to proceed as a cause of poor documentation and overview, and lastly inadequate teamwork from having unsafe and suboptimal collaboration features.

References

Baltes, S., & Diehl, S. (2018). Towards a Theory of Software Development Expertise. *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering - ESEC/FSE 2018*, 187–200. https://doi.org/10.1145/3236024.3236061

Beck, S. E., & Manuel, K. (2008). *Practical research methods for librarians and information professionals*. New York, NY: Neal-Schuman.

Beecham, S., Baddoo, N., Hall, T., Robinson, H., & Sharp, H. (2008). Motivation in Software Engineering: A systematic literature review. *Information and Software Technology*, *50*(9–10), 860–878. https://doi.org/10.1016/j.infsof.2007.09.004

Bell, J., & Waters, S. (2016). *Introduktion till forskningsmetodik* (5., [updated] ed.). Lund: Studentlitteratur AB.

Besker, T., Ghanbari, H., Martini, A., & Bosch, J. (2020). The influence of Technical Debt on software developer morale. *Journal of Systems and Software*, *167*, 110586. https://doi.org/10.1016/j.jss.2020.110586

Besker, T., Martini, A., Bosch, J. (2018). Managing architectural technical debt: a unified model and systematic literature review. *Journal of Systems and Software*, 135, 1–16 Supplement C.

Bryman, A., 2011. *Samhällsvetenskapliga metoder*. 2 ed. Malmö: Liber.

CompTIA (2019) IT INDUSTRY OUTLOOK. Retrieved from https://comptiacdn.azureedge.net/ webcontent/docs/default-source/research-reports/comptia-it-industry-outlook-2020.pdf?sfvrsn=8869ad68_0

Dourish, P. (2003). The appropriation of interactive technologies: Some lessons from placeless documents. *Computer Supported Cooperative Work*, 12, 465–490. doi:10.1023/ A:1026149119426

Fagerholm, F. (2015). *Software Developer Experience: Case Studies in Lean-Agile and Open Source Environments* (Doctoral dissertation, Ph. D. Dissertation. Department of Computer Science, University of Helsinki. Series of Publications A, Report A-2015-7).

Fagerholm, F., & Munch, J. (2012). Developer experience: Concept and definition. *2012 International Conference on Software and System Process (ICSSP)*, 73–77. https://doi.org/10.1109/ICSSP.2012.6225984

Fontão, A., Bonifácio, B., Santos, R. P. dos, & Dias-Neto, A. C. (2018). Mobile Application Development Training in Mobile Software Ecosystem: Investigating the Developer eXperience. *Proceedings of the 17th Brazilian Symposium on Software Quality - SBQS*, 160–169. https://doi.org/10.1145/3275245.3275262

Fontão, A., Ekwoge, O. M., Santos, R., & Dias-Neto, A. C. (2017). Facing up the primary emotions in Mobile Software Ecosystems from Developer Experience. *Proceedings of the 2nd Workshop on Social, Human, and Economic Aspects of Software - WASHES '17*, 5–11. https://doi.org/10.1145/3098322.3098325

França, C., Sharp, H., & da Silva, F. Q. B. (2014). Motivated software engineers are engaged and focused, while satisfied ones are happy. *Proceedings of the 8th ACM/IEEE*

International Symposium on Empirical Software Engineering and Measurement - ESEM '14, 1–8. https://doi.org/10.1145/2652524.2652545

Graziotin, D., Fagerholm, F., Wang, X., & Abrahamsson, P. (2017a). Consequences of Unhappiness While Developing Software. *2017 IEEE/ACM 2nd International Workshop on Emotion Awareness in Software Engineering (SEmotion)*, 42–47. https://doi.org/10.1109/SEmotion.2017.5

Graziotin, D., Fagerholm, F., Wang, X., & Abrahamsson, P. (2017b). Unhappy Developers: Bad for Themselves, Bad for Process, and Bad for Software Product. *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, 362–364. https://doi.org/10.1109/ICSE-C.2017.104

Graziotin, D., Wang, X., & Abrahamsson, P. (2014). Happy software developers solve problems better: Psychological measurements in empirical software engineering. *PeerJ*, *2*, e289. https://doi.org/10.7717/peerj.289

Greig, J. (2020). COVID-19 triggering a massive shift in adoption of low-code platforms. TechRepublic. Retrieved from https://www.techrepublic.com/article/covid-19-triggering-a-massive-shift-in-adoption-of-low-code-platforms/

Hassenzahl, M., & Tractinsky, N. (2006). User experience—A research agenda. *Behaviour & Information Technology*, *25*(2), 91–97. https://doi.org/10.1080/01449290500330331

International Organization for Standardization (2019). Ergonomics of human-system interaction — Part 210: Human-centred design for interactive systems (ISO 9241-210:2019). Retrieved from https://www.iso.org/obp/ui/#iso:std:iso:9241:-210:ed-2:v1:en

Kärpänoja, P., Virtanen, A., Lehtonen, T., & Mikkonen, T. (2016). Exploring Peopleware in Continuous Delivery. *Proceedings of the Scientific Workshop Proceedings of XP2016 on - XP '16 Workshops*, 1–5. https://doi.org/10.1145/2962695.2962708

Koksal, I. (2020, April 29). The Rise Of Low-Code App Development. Forbes. https://www.forbes.com/sites/ilkerkoksal/2020/04/29/the-rise-of-low-code-app-development/

Kuusinen, K. (2015). Software Developers as Users: Developer Experience of a Cross-Platform Integrated Development Environment. In P. Abrahamsson, L. Corral, M. Oivo, & B. Russo (Eds.), *Product-Focused Software Process Improvement* (Vol. 9459, pp. 546–552). Springer International Publishing. https://doi.org/10.1007/978-3-319-26844-6_40

Kuusinen, K., Petrie, H., Fagerholm, F., & Mikkonen, T. (2016). Flow, Intrinsic Motivation, and Developer Experience in Software Engineering. In H. Sharp & T. Hall (Eds.), *Agile Processes, in Software Engineering, and Extreme Programming* (Vol. 251, pp. 104–117). Springer International Publishing. https://doi.org/10.1007/978-3-319-33515-5_9

Law, E. L. C., Roto, V., Hassenzahl, M., Vermeeren, A. P., & Kort, J. (2009, April). Understanding, scoping and defining user experience: a survey approach. In *Proceedings of the SIGCHI conference on human factors in computing systems* (pp. 719-728).

Mcconnell-Henry, T., James, A., Chapman, Y., & Francis, K. (2009). Researching with People You Know: Issues in Interviewing. *Contemporary Nurse : a Journal for the Australian Nursing Profession*, *34*(1), 2–9.

Misirli, A. T., Verner, J., Markkula, J., & Oivo, M. (2014). A survey on project factors that motivate Finnish software engineers. *2014 IEEE Eighth International Conference on Research Challenges in Information Science (RCIS)*, 1–9. https://doi.org/10.1109/RCIS.2014.6861052

Morris, A. (2017). The rise of low-code development. Linx Software. Retrieved from https://linx.software/the-rise-of-low-code-development/

Multisoft. (2020). Automatisera med marknadens mest effektiva Low-code plattform. Multisoft®. Retrieved from https://www.multisoft.se/softadmin/

Nylund, A. (2020). *A multivocal literature review on developer experience*. (Unpublished master's thesis, Aalto University, School of Science, Esbo, Finland) http://urn.fi/URN:NBN:fi:aalto-202003222600

Oliveira, R., & França, C. (2019). Agile Practices and Motivation: A quantitative study with Brazilian software developers. *Proceedings of the Evaluation and Assessment on Software Engineering*, 365–368. https://doi.org/10.1145/3319008.3319714

Palviainen, J., Kilamo, T., Koskinen, J., Lautamäki, J., Mikkonen, T., & Nieminen, A. (2015). Design framework enhancing developer experience in collaborative coding environment. Proceedings of the 30th Annual ACM Symposium on Applied Computing - SAC '15, 149–156. https://doi.org/10.1145/2695664.2695746

Sanchis, R., García-Perales, Ó., Fraile, F., & Poler, R. (2019). Low-Code as Enabler of Digital Transformation in Manufacturing Industry. *Applied Sciences*, *10*(1), 12. https://doi.org/10.3390/app10010012

SHRM. (2019). THE GLOBAL SKILLS SHORTAGE - *Bridging the Talent Gap with Education, Training and Sourcing*. https://www.shrm.org/hr-today/trends-and-forecasting/research-and-surveys/Documents/SHRM%20Skills%20Gap%202019.pdf

Taherdoost, H. (2016). Sampling Methods in Research Methodology; How to Choose a Sampling Technique for Research. *SSRN Electronic Journal*. https://doi.org/10.2139/ssrn.3205035

Tchounikine, P. (2017). Designing for Appropriation: A Theoretical Account. *Human–Computer Interaction*, 32(4), 155–195. https://doi.org/10.1080/07370024.2016.1203263

Tisi, M., Mottu, J.-M., Kolovos, D. S., de Lara, J., Guerra, E., Ruscio, D. D., Pierantonio, A., & Wimmer, M. (2019). *Lowcomote: Training the Next Generation of Experts in Scalable Low-Code Engineering Platforms*. 6.

Vetenskapsrådet, 2002. *Forskningsetiska principer*, Stockholm: Vetenskapsrådet.

Waszkowski, R. (2019). Low-code platform for automating business processes in manufacturing. *IFAC-PapersOnLine*, *52*(10), 376–381. https://doi.org/10.1016/j.ifacol.2019.10.060

Yin, R. K. (2011). Qualitative Research from Start to Finish. New York: Guilford.