Niko Tuominen

EXPLORATION OF THE DIFFERENCES IN DEVELOPER EXPERIENCE OF AN IOT LOW-CODE DEVELOPMENT PLATFORM BETWEEN CITIZEN DEVELOPERS AND PROFESSIONAL DEVELOPERS

UNIVERSITY OF JYVÄSKYLÄ
INFORMATION SYSTEMS
2023

ABSTRACT

Tuominen, Niko
Exploration of the differences in Developer Experience of an IoT Low-Code Development Platform between Citizen Developers and Professional Developers
University of Jyväskylä, 2023
Information Systems, Master's Thesis
Supervisor(s): Halttunen, Veikko

Low- and no-code development platforms are software platforms that enable the creation of applications through graphical user interfaces rather than through traditional programming. This enables software development for the so called 'citizen developers' – that is, people without a software developer background – and thereby helps answer the current shortage of highly skilled software developers. This thesis explores and compares through nine qualitative interviews the differences in the perception of the developer experience of an IoT low-code development platform held by both citizen developers as well as users with a professional software developer background. Based on an analysis of the interviews, it was discovered that while there are similarities between the two groups' experiences, there are also differences. The three main differences between the two user groups' experiences were how the users' sense of the platform's ease of use developed over time, how smooth or rough the use felt, and how the platform was experienced in terms of limiting or enabling creativity. The causes for negative developer experience are evaluated and several feature improvement ideas for the case platform were formulated based on the interviews.

Keywords: Low-Code Development Platform, Citizen Developer, Developer Experience, Internet of Things

TIIVISTELMÄ

Tuominen, Niko
IoT low-code kehitysalustan kehittäjäkokemuksen erot ammatti- ja kansalaiskehittäjien välillä
Jyväskylän Yliopisto, 2023
Tietojärjestelmätiede, pro gradu -tutkielma
Ohjaaja(t): Halttunen, Veikko

Low- ja no-code kehitysalustat ovat alustoja, jotka mahdollistavat ohjelmistokehityksen graafisten käyttöliittymien kautta perinteisen koodin kirjoittamisen sijaan. Tämä tekee sovellusten kehittämisen mahdolliseksi myös niin sanotuille kansalaiskehittäjille – eli kehittäjille ilman ohjelmistokehittäjätaustaa – sekä auttaa täten yrityksiä vastaamaan vallitsevaan pulaan ohjelmistokehittäjistä. Tämä tutkielma vertailee yhdeksän kvalitatiivisen haastattelun kautta kehittäjäkokemuksen eroja IoT low-code kehitysalustan käytössä ammattimaisten ohjelmistokehittäjien sekä kansalaiskehittäjien välillä. Tulosten analyysin perusteella huomattiin, että vaikka kokemuksissa on paljon samaa, on niissä myös eroavaisuuksia. Kolme pääeroavaisuutta oli siinä, kuinka alustan helppokäyttöisyyden tunne muuttui käyttäjillä ajan myötä, kuinka sulavalta käyttö tuntui, sekä kuinka paljon alustan koettiin rajoittavan tai mahdollistavan käyttäjän luovuutta. Negatiivista kehittäjäkokemusta aiheuttavien seikkojen syitä analysoitiin ja haastattelujen pohjalta alustan tarjoajalle esitettiin jatkokehitysehdotuksia.

Avainsanat: Low-code kehitysalusta, kansalaiskehittäjä, kehittäjäkokemus, esineiden internet

FIGURES

TABLES

TABLE OF CONTENTS

1 INTRODUCTION

Companies are faced with an ever growing need to stay agile, innovative, and to rapidly bring out new services to exploit new business models - all utilizing IT. Leveraging digitalization and digital transformation is essential for staying on top of the rapidly changing world. (Elshan, Dickhaut & Ebel, 2023) This shared need has resulted in an increased demand for programmers, developers, and IT specialists that higher education institutions have not been able to meet, which in turn shows itself to businesses as a shortage of ICT professionals. (Alsaadi et al., 2021)

Low-code development platforms (LCDPs) aim to tackle this problem by both increasing the speed of software development for professional developers and by enabling it for the so-called citizen developers. (Rokis & Kirikova, 2023; Rafi et al., 2022) This is done by providing the user with means to create applications through the use of graphical user interfaces as opposed to traditional programming, removing the need for actual code-writing. (Sahay et al., 2020) Citizen developer in this thesis refers to any application creators who do not have a prior professional or educational background in software development.

LCDPs are part of a larger trend of technology democratization that diminishes the gap between subject specialists and IT departments, giving companies a much larger talent pool of employees capable of creating solutions. In addition to helping answer the shortage of software developers, this also lets the subject matter experts design software for themselves, driving innovation. Beyond this, since in low-code projects the role of developer and commissioner is mixed, communication effort and opportunities for misunderstandings are reduced. (Elshan et al., 2023)

The use of LCDPs present itself to businesses with many potential benefits, such as the ability to keep solution development in-house, making it more rapid and less costly, reducing complexity and increasing maintainability, and resulting in apps that can be quickly confirmed to answer the actual customer need.

(Sanchis et al., 2020) Other advantages include increased flexibility, modularity, and cost-effectiveness. (Cai et al., 2022)

However, while multiple studies have shown the use of LCDPs can accelerate the development cycle and reduce cost (Rokis & Kirikova, 2023) not all professional developers are so ecstatic about adopting their use over traditional programming. (Alsaadi et al., 2021) Elshan et al. (2023) note that it can be especially difficult to convince IT professionals to adopt low-code, as developers often consider themselves as painters who don't want some tool to automatically generate their piece.

Software Engineering (SE) is the activity through which IT systems are created, typically using tools such as integrated development environments (IDEs). While the developers themselves, as well as the nature of development work itself, has been extensively studied, the IDEs, while playing an important role in the creation of software, have received less attention. (Kuusinen et al., 2016)

Developer experience (DX) is a concept that addresses the way developers perceive the development infrastructure, how they feel about their work, how they see the value of their contribution. (Fagerholm & Münch, 2012)

This thesis exists in the cross-section of these concepts, investigating the different experiences of software engineering using an LCDP, held by both traditional code-writing professional software developers and citizen developers. More importantly, these two user groups' experiences are compared against each other to see what kinds of differences there may be.

Low-code platforms are a relatively recent phenomenon that has only in the past few years started receiving significant academic attention in the form of publications. Large portion of the existing research has been published in workshops and conferences (Bucaioni, Cicchetti & Ciccozzi, 2022), and most of this literature focuses on the technical rather than social and human aspects (Prinz, Huber & Rentrop, 2021). Alsaadi et al. (2021) have called for more studies and assessments to be done on how to provide solutions to the challenges and issues developers are facing in using LCDPs. The DX of LCDPs has previously been explored by a small number of mainly other thesis publications.

To the best of my knowledge the comparison of DX between citizen developers and professional developers in using the same platform is novel and may bring new insight into the different perspectives of these two user groups. The study focuses itself on the following research questions, inspired by both the literature already referenced above, as well as practical needs arising from the industry.

RQ1: “What kind of differences in the developer experience of an IoT low-code development platform exist between users with a software developer background and those without?”
RQ2: “How can any possibly negative experiences be best addressed?”

The research questions and how they are posed make two assumptions. The first assumes there to be differences between the two user groups and how they consider the DX of the case LCDP. The second expectation is that professional software developers may not be as excited or positive about using an LCDP as citizen developers are. Therefore, the thesis formally uses the following hypothesis.

H1: “Professional software developers experience using low-code development platforms differently compared to citizen developers”
H2: “Professional software developers are less eager to use low-code development platforms compared to citizen developers”

These research questions and hypothesis are explored by conducting and analysing the results of nine qualitative interviews split to include five professional software developers and four citizen developers.

This thesis was commissioned by a Finnish software company Wapice Ltd., but the topic for the research was brought up by the author independently. Wapice Ltd. is a Finnish software company established in 1999 that has been actively developing its own IoT platform since 2005. From 2011 onwards the platform has included elements of graphical widget-based application building and in 2015 it, after having become entirely web-based, was rebranded as IoT-TICKET. (Wapice, 2017) According to Bock and Frank (2021), this is quite typical of the products that now fall under the “low-code” umbrella – they have often existed well before the term itself was coined, but under different names, such as that of “rapid application development platforms”.

The thesis is structured as follows: First, based on a literature review the theoretical basis and practical background of the thesis is explained in the second chapter. The third chapter describes the chosen research methodology of qualitative interviews and justifies its use. The fourth chapter describes and analyses the results of the interviews on their own, while the fifth considers these results against the existing literature. A summary of the thesis is provided in the sixth chapter, along with suggestions for future research.

2 BACKGROUND AND LITERATURE REVIEW

This chapter is divided into two main parts. First, the key concepts of low-code development platforms and developer experience are defined, and the way they are presented in the existing research literature is analysed. Then, in the second part, to take into account the Industrial Internet of Things nature of the case platform, this topic is also explored and a description of the platform itself is provided in an effort to help give context to the research results covered in later chapters.

2.1 Theory background

This sub-chapter will cover the theoretical background of the thesis. It is based on a semi-systematic literature review conducted using Google Scholar. Articles were searched with the query "Low-code platform" OR "No-code platform" OR "Low-code development platform" OR "LCDP" AND "Citizen developer" AND "Developer Experience". This yielded only 11 results, of which one was excluded due to being in German. A snowballing method was used to expand the search to other relevant articles focusing on fewer terms.

2.1.1 Low- and no-code platform (LCDP)

There are many overlapping but slightly different definitions for what low-code development platforms are. It seems to be generally accepted that the term was initially coined in 2014 by the market research company Forrester Research. (Di Ruscio et al., 2022; Bucaioni et al., 2022; Alyousef, 2021) In this initial definition, Forrester defined low-code platforms as "*Platforms that enable rapid delivery of business applications with a minimum of hand-coding and minimal upfront investment in setup, training, and deployment*". (Richardson & Rymer, 2016) Over time Forrester's definition has evolved to include and highlight the WYSIWYG, or what-you-see-is-what-you-get, aspects LCDPs focusing on using visual, declarative techniques as opposed to programming to create applications (Di Ruscio et al., 2022).

Gartner, another significant market research company, introduced the concept of low-code application platforms, or LCAPs, few years later in 2016 (Di Ruscio et al., 2022), but it seems that in literature the two terms are used interchangeably, with LCDP being the slightly more prevalent term. For the purposes of this study, LCDP and LCAP are considered interchangeable.

It is worth noting that Gartner considers the term no-code application platform, or NCAP, to be primarily used as a marketing and positioning statement and opts to include these platforms within the umbrella of LCAPs. (Vincent et al., 2019) This sentiment seems to be shared by other researchers (Bucaioni et al., 2022; Prinz et al., 2021), so in the context of this study the concepts of no-code development platforms or no-code application platforms will be included under LCDP and only the last term will be used.

In academia, Sahay et al. (2021) present what seems to be the most in-depth and complete definition of LCDPs: LCDPs are cloud-based software platforms that enable developers with different domain knowledge and of varying technical expertise to develop production ready applications. The applications are developed utilizing MDE principles and taking advantage of cloud infrastructures, automatic code generation, and declarative high level graphical abstractions. The platforms are provided as Platform-as-a-Service (PaaS) and ensure effective and efficient development, deployment, and maintenance of the desired applications. The advantage of using LCDPs provides is not only that they enable creation of software to citizen developers, but that they allow full-stack developers to focus on business logic of the applications rather than dealing with setting up needed infrastructures, managing data integrity across different environments and enhancing the robustness of the system. (Sahay et al., 2021) Rafi et al. (2022) also note that LCDPs can help novice and aspiring coders to write code with reduced complexity, allowing the more experienced programmers to focus on core project areas and other productive project activities.

Analysing posts and comments posted on Stack Overflow and Reddit to gain an understanding of how practitioners themselves perceive low code development, Alyousef (2021) found the most common descriptors for low-code to be "requiring low coding effort" and including "drag and drop" or "visual programming". Other notable descriptions included "using pre-designed templates", being "non-professional programmer friendly", and using "WYSIWYG" (what-you-see-is-what-you-get) principles. (Alyousef, 2021)

Evaluating eight prominent LCDPs, Sahay et al. (2021) provide a breakdown of a typical architecture of an LCDPs by dividing them into four architectural layers: an application layer, a service integration layer, a data integration layer, and the deployment layer as shown on Figure 1. In the first layer, applications are defined in an application modeler that uses modelling constructs and abstractions such as widgets and connectors. A service integration layer facilitates connecting different services by using APIs and authentication mechanisms, while data integration layer ensures data integrity of the data from different sources. The service and data integration layers can be managed by the backend without needing user intervention, although developers can be provided with external APIs. Finally, deployment layer handles the containerization and orchestration of the applications in either cloud infrastructures or in an on-premise environment. (Sahay et al., 2021)

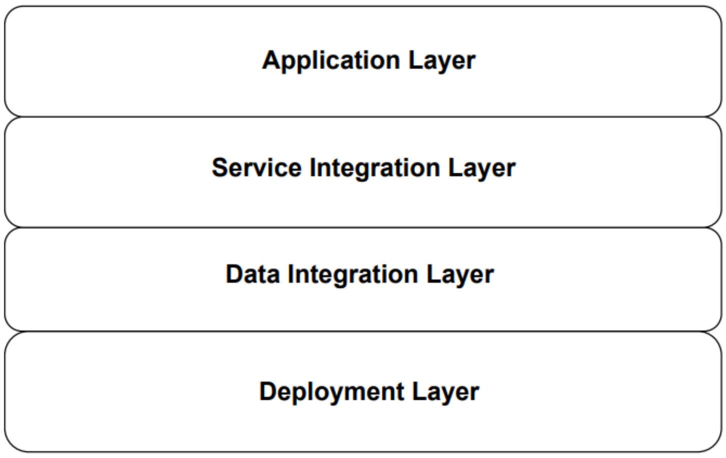


Figure 1 Layered architecture of LCDPs (Sahay et al., 2021)

Sahay et al. (2021) summarize the development process in LCDPs as consisting of the following stages 1. the data modelling, 2. user interface definition, 3. business logic rule and workflow specification, 4. integration of external services, and finally 5. application deployment. While agreeing and directly referencing these stages, Alamin et al. (2021) add that applications can be created using LCDPs in two directions - either UI first followed by data design, or data first and then UI second. Regardless of which approach is used, agile methodologies can be applied in a circular process where customer satisfaction is sought by continuous incremental delivery. (Alamin et al., 2021)

Waszkowski (2019) states that the approaches that created low-code programming are model driven software development approach, rapid application development, automatic code generation and visual programming. This is echoed by Sahay et al. (2021) who state that model driven engineering, or MDE, principles are at the heart of LCDPs. Bucaioni et al. (2022) also support this notion, their analysis showing MDE as the most mentioned core technology in both peer-reviewed and grey low-code literature.

MDE is a software development paradigm that increases the abstraction level of the software development process by allowing the development of systems using models defined with concepts that are closer to the problem domain than the underlying implementation technology. The purpose of this is to increase productivity and reduce time to market, as well make the solutions easier to specify, understand and maintain. While what a model is in MDE is not rigidly defined, they can be considered a communication of ideas between computers and humans that are manipulatable in an automated way. Models and artifacts, the things defined within a model, are themselves defined by metamodels. Much like a computer program would conform to the grammar of the language's syntax it is written in, models must conform to their metamodels. Models can be transformed from one model to another using model transformations. (Di Ruscio et al., 2012) Further, executable software can be generated from these transformations. (Bucaioni et al., 2022)

The level to which MDE and low-code are considered overlapping varies depending on the author and no clear consensus seems to exist. Buscaioni et al. (2022) take the stance of positioning low-code development as "a set of methods and/or tools in the context of a broader methodology, often being identified as model-driven engineering.". They further note that LCD could even be considered as a "maturation step of MDE" in terms of usability and flexibility. Cabot (2020) goes as far as to say that the two are synonymous, while Di Ruscio et al. (2022) counterpoints this by highlighting that that not all MDE approaches seek reduction of the amount of code and not all low-code approaches are model-driven.

Di Ruscio et al. (2022) further note differences between MDE and LCDPs in their platforms, users, and domain, as highlighted in Table 1.

Table 1 Differences between low-code development and model driven engineering, based on Di Ruscio et al. (2022)

Low-code development	**Model Driven Engineering**
Typically cloud based, PaaS	Typically desktop based
Users typically Citizen Developers and domain experts	Users typically professional software developers
Typically domain agnostic, although also domain specific to business applications, IoT, machine learning and assistants	Typical targets technical domains such as automotive, power engineering, and cyber physical systems at large.

It seems overall that the MDE versus low-code discourse has a lot of colourful subtexts depending on the camp one approaches it from. On one hand the tone of some of the papers gives the impression that there might be an element of jealousy among MDE practitioners towards the current interest towards and hype around low-code, but on the other it is also acknowledged that the same might be an opportunity for applying existing knowledge and cross-pollination, as admitted by Cabot (2020) and Di Ruscio et al. (2022).

Prinz et al. (2021) note that most of the published LCDP-related literature has focused on technology related topics rather than the social and human aspects. Bucaioni et al. (2022) note that so far most of the studies on low-code have been published in either workshops or conferences, with the number of journal publications still lagging. This highlights the still early stage of low-code focused research, something that this study contributes towards addressing. In addition, there has been a significant year-on-year increase in the number of publications and even during the writing process of this thesis several new works have been published.

2.1.2 Developer experience (DX)

While in the context of information systems and software development the concept of User Experience, or UX for short, has been long established and covered, significantly less attention has been given to the experiences of the developers developing that software. Influenced by the UX concept, Fagerholm and Münch (2012) proposed the initial definition and concept of developer experience. Despite their original acronym for developer experience being DEx, DX seems to be the more commonly adopted form. For this reason, the latter is used in this thesis.

According to this initial definition, DX consists of experiences relating to both the artifacts and activities that developers encounter as part of creating software. Fagerholm and Münch divided the concept of DX into three subareas: cognition, affect and conation. Cognition addresses how developers perceive the development infrastructure, affect how they feel about their work, and conation how they see the value of their contribution. Developers are broadly defined as anyone engaged in the activity of developing software - not necessarily only professional programmers. (Fagerholm & Münch, 2012)

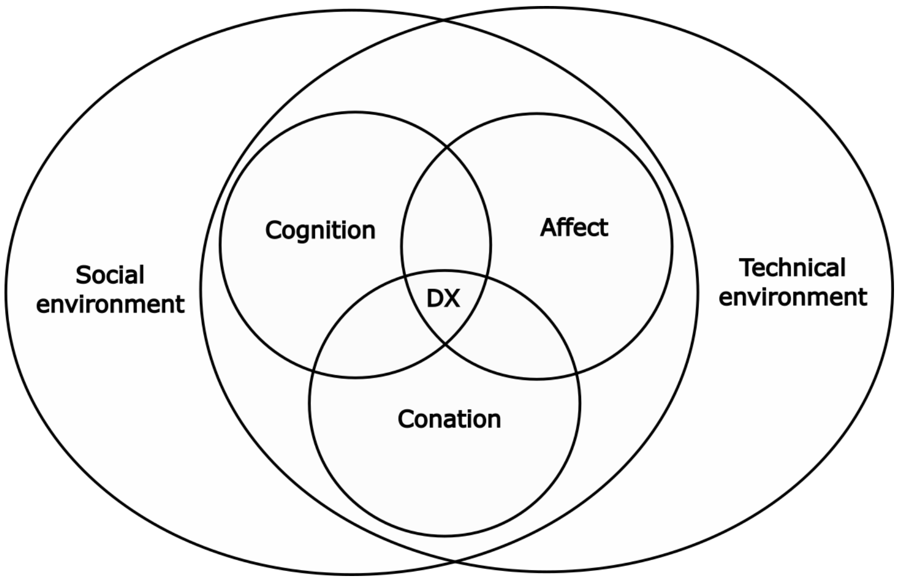


Figure 2 Fagerholm's (2015) concept of developer experience (DX) and its social and technical environment

Fagerholm (2015) expands on this initial concept with the inclusion of the two different environments software development is done in: a social and technical environment. This is portrayed in Figure 2. The social environment is based on social psychological mechanisms such as beliefs, norms, values and group formation and identity. The technical environment consists of artefacts such as programming languages, the written code itself, the tools the code is written in, as well as plans, diagrams, and processes. (Fagerholm, 2015)

Fagerholm (2015) also provides an approach in the form of a theoretical model that can be used to frame observations into DX. In it, DX has seven aspects. First aspect is the *experience object*, or what exactly is being experienced. This could be something concrete such as a technical artefact, or something more abstract such as a process. Second aspect is *experience formation*, which describes how the experience is formed as the experience object is encountered. The third aspect, *experience influencers* are external and internal factors that moderate experience formation. Fourth is *experience content*, which is the subjective evaluation of the formed experience, containing for example judgements of the positivity or negativity of the experience. Fifth factor, *experience progression*, describes how the experience evolves over time. Sixth, the negative or positive experiences can lead to *behaviour outcomes*, that is changes in the developers' overt behaviour. And ultimately seventh, *object outcomes* are the changes that the behaviour outcomes may lead to in the experience object itself. These aspects are summarized in Table 2. Fagerholm (2015) uses an integrated development environment as an example of what this approach could be used to evaluate.

Table 2 Aspects of DX (Fagerholm, 2015)

Aspect	**Description**
Experience object	What is experienced
Experience formation	How the experience is formed
Experience influencers	Factors that influence the experience
Experience content	The parts or elements of the experience
Experience progression	How the experience changes over time
Behaviour outcome	How the experience leads to or moderates behaviour
Object outcome	How the experience or resulting behaviour leads to or moderates changes in one or more artefact or phenomenon

Other approaches to measuring DX have been presented by Lee and Pan (2021), who took the three subconstructs presented by Fagerholm & Münch (2012) and tested their validity in measuring the DX of a deep learning platform. In their work, cognition was defined as referring to attention, memory, and problem-solving related factors. Affection concerns the developer's feelings and emotions, and conation consists of impulse, motivation, and desire. They conclude that DX, at least of a deep learning platform, can be measured based on the developer's feelings of value perceived from the platform, their positive or negative affection, and willingness to use it. (Lee & Pan, 2021)

Lee and Pan (2021) note that understanding the relationship between the platform and developers using it can help us predict whether the platform can satisfy developers and to ensure its usability and functionality. This is echoed by Nylund (2020), who writes that understanding the different aspects impacting DX is beneficial when designing a software development platform or environment, as the platform can then be aligned with the developers using it. Further, Graziotin et al. (2017) note that unhappiness of software developers can lead to low mental performance, motivation, and productivity, as well as delays in the project or cutting of corners. Unhappiness can also lead to mental health issues, such as low self-esteem, anxiety, stress, burnout, and depression. (Graziotin et al., 2017)

While developer experience as a term often refers to the kind of DX discussed in this thesis, this is not always the case. Brasil-Silva and Selvy Siqueira (2022) presented a systematic mapping to analysing the metrics in determining software developer experience, but their study focused on metrics determining the level of experience in the context of seniority of the software developers. Searching for articles just on developer experience yielded more results interpreting the term this way, which shows perhaps that developer experience as a concept is not entirely established and there can be ambiguity in what the term sometimes refers to. This was also noted by Nylund (2020). In the context of this thesis, the concept of DX is used for talking about the experiencing of software development as an activity and its related artifacts.

2.1.3 DX of LCDPs

As shown by the lack of research articles resulting from the initial query, research into the DX of LCDPs is still scarce. However, a few related master's theses have been published during just the past couple of years. It's perhaps worth noting that two of these have been supervised by Fagerholm, who can be seen as one of the key contributors to the DX literature so far.

In one of these theses, Kermanchi (2022) compared in an experiment the difference in developer experience of developers between using LCDPs and traditional programming. They found that the amount of prior programming experience the developer had had a strong influence on the developer's performance, experiences, and preferences. They highlighted that their participants saw scalability and imposed limitations of the LCDPs as challenge the platforms need to address. Salesforce's Flow Builder and APEX programming language were used in the test and all participants came from IT backgrounds. (Kermanchi, 2022)

Alyousef (2021) explored through qualitative interviews the challenges developers faced in using LCDPs. They interviewed both citizen developers and professional developers experienced with different LCDPs, allowing them to compare the challenges they faced. They found that citizen developers, lacking experience and backgrounds in software development, had trouble getting up to speed and becoming productive in implementing solutions with the LCDPs. They also reported that the citizen developers faced challenges when trying to extend their applications using traditional development technologies. Conversely, they found that professional developers can switch to using LCDPs easily. Two of the three professional developers interviewed stated they thought low-code platforms should focus more on professional developers, and that professional developers' experience and the low-code tools are a powerful combination. Further research was called into different platforms to see if findings are similar, and to also investigate further the role of citizen developers and how it could be improved. (Alyousef. 2021)

Hallberg (2021) conducted an autoethnography on using two different LCDPs, Mendix and EHR Studio, in order to create a web application prototype for collecting patient-generated health data. While they only gathered data about their own experiences, limiting the potential generalizability of the results, they discovered both positive and negative contributing aspects of LCDPs towards DX. Positive aspects were quick and easy implementation. Negative aspects had to do especially with the completeness of the platform's features: being unable to complete their task without having to program significant parts themself. They highlight the platform's impact on creativity as both an inspiring and a limiting factor. (Hallberg, 2021)

Gao (2021) researched the episodic experience of DX of LCDPs. Episodic experience is the experiences that arises over minutes or hours and can be contrasted with cumulative experience, which is experience gained over a long period of time. Gao developed first a preliminary questionnaire based on a collection and consolidation of different metrics from various existing surveys, followed by a delphi study to refine them into the final questionnaire that was given to participants after they had completed a task-oriented test. Analysis of the questionnaire results indicated that participants with programming experience did rate their experience in completing the research task differently to those that did not. However, they noted that not all prior programming experience is necessarily helpful and programming experience alone may in fact be too vague to use as a metric. They suggested further studies should differentiate and consider front-end development experience specifically, as a closer equivalent to the workflow of LCDPs. Interestingly, prior LCDP or design experience did not consistently predict performance in completing the tasks. In discussing the contribution of their work, Gao noted that there exists debate about whether quantitative or qualitative feedback is more important in informing UX and DX design and development decisions. Gao's research explored the quantitative approach.

Finally, Dahlberg (2020) studied the DX of the LCDP Softadmin by interviewing software developers and a project leader working with the platform. They found that developers felt positively about the increased productivity and opportunities for collaborating with the customer enabled by the LCDP. Negative experiences resulted from platform constraining their problem-solving ability and adding personal touch to layouts being limited. However, it was also noted that certain constraints on creativity can be acceptable if justified by increased productivity. Insight into the bigger picture within the LCDP was seen as a challenge along with the difficulty of collaboration while working with other developers. Importance of proper documentation and support from the platform provider was highlighted, as well as the potential benefits of a rich learning process.

2.2 Practical background

The second part of this chapter focuses on the practical context of the case platform and then the platform itself. First, a brief history of the concept of IoT and its history is covered and then a description of the case LCDP IoT-TICKET is provided. The topic of IoT is discussed because that is the context in which the users are using the case LCDP in this study. A description of the case platform

IoT-TICKET is provided so that the results of the study can be more easily understood in their context.

2.2.1 Internet of Things (IoT) platforms

Even before the term was coined, things have long been connected to the Internet. What constitutes the first IoT device depends on how the term is defined, but significant contenders for the title include a 1982 Coke soda machine in the Carnegie Mellon University and a 1990 toaster that could be turned on and off over the Internet by John Romkey. (Suresh et al., 2014)

The term Internet of Things was coined initially by Kevin Ashton in a presentation to Procter & Gamble in 1999, just before the change of the millennium. In his presentation Ashton was discussing combining the ideas of RFID technology and Internet in supply chains. (Ashton, 2009; Chin, Callaghan & Allouch, 2019) Ashton's vision for this combination of technologies was of a world where computers are "using data they gathered without any help from us [humans]" leading to more accurate tracking, reduced waste, loss, and cost. (Ashton, 2009) Even if the technology landscape has transformed significantly since then, the underlying goal of utilizing IoT to optimize processes has well stood the test of time.

IoT is a key driving technology behind digitalization. Yoo, Henfridsson and Lyytinen (2010) describe how hardware miniaturization, microprocessors, improvements in memory, broadband communication, and battery management technologies have led to digitalization of the key functions of many traditional products. Building on this, Wortmann and Flüchter (2015) write that IoT solutions are enabling creation of new IT-based digital services by combining physical products with additional software and hardware. These digital services can create value to their users either by enhancing the products primary functions, or by networking multiple related products together. This is in line with Porter and Heppelmann (2014), who call these IoT devices "smart, connected products". They go on to group the capabilities of these Smart Connected Products enabled by IoT into four areas: monitoring, control, optimization, and autonomy. These areas build on top of the previous and are foundational to the next.

As more and more devices gather increasingly complex data about themselves and the world around them, the tools that are used to manage and analyse this data become increasingly important. This requires a stack of technology from hardware and software on the device itself, to a connectivity layer, and ultimately a software cloud layer. (Porter & Heppelmann, 2014)

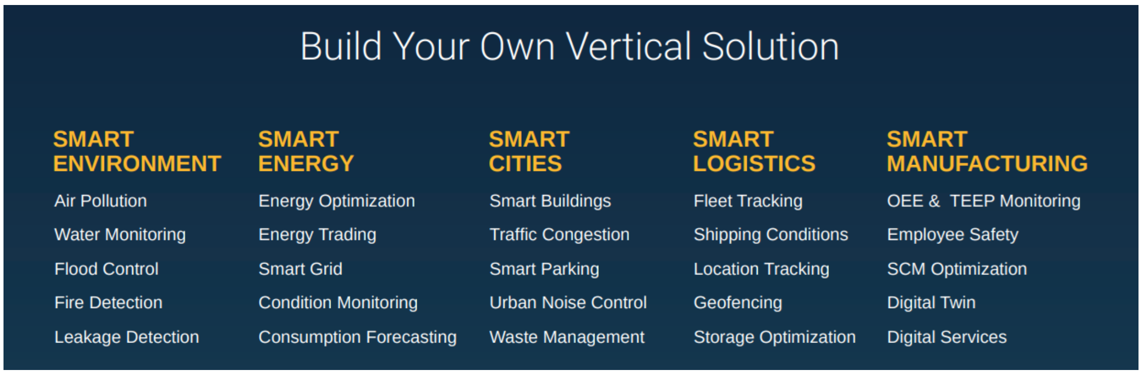
IoT platforms are software products that come equipped with features and functionalities with which IoT applications themselves can be built. A multitude of IoT platforms exists to address different application areas. Platforms also stand out from each other in which parts of the technology stack they focus on and what functionalities they offer. (Wortmann & Flüchter, 2015)

2.2.2 IoT-TICKET

IoT-TICKET is an IoT platform developed and provided by Finnish software company Wapice Ltd. Its development started in 2005 under the name of Wapice Remote Management, or WRM for short. From 2011 onwards, the platform has included graphical widget-based "Flow-programming", making it an LCDP even if the term itself had not yet been coined. At the time, labels such as rapid application development platform were used instead. After a major update in 2015, which included updating the platform to be delivered entirely through the use of browser-technologies such as HTML5, it was rebranded as IoT-TICKET. (Wapice, 2019)

At the time of writing IoT-TICKET Generation 4 is the latest major version, published in late 2021. Some of the platform's customers, also of those interviewed in this study, are still also using IoT-TICKET 3.

Wapice positions IoT-TICKET as a "generic" IoT platform in that it can be used to create solutions to a multitude of different application areas ranging from smart environment, energy, cities, logistics, and manufacturing fields. Different use cases within these application areas are shown on Figure 3. (Wapice, n.d.-a)



Build Your Own Vertical Solution

SMART ENVIRONMENT	SMART ENERGY	SMART CITIES	SMART LOGISTICS	SMART MANUFACTURING
Air Pollution	Energy Optimization	Smart Buildings	Fleet Tracking	OEE & TEEP Monitoring
Water Monitoring	Energy Trading	Traffic Congestion	Shipping Conditions	Employee Safety
Flood Control	Smart Grid	Smart Parking	Location Tracking	SCM Optimization
Fire Detection	Condition Monitoring	Urban Noise Control	Geofencing	Digital Twin
Leakage Detection	Consumption Forecasting	Waste Management	Storage Optimization	Digital Services

Figure 3 IoT-TICKET Application areas, according to Wapice (n.d.-a)

IoT-TICKET is used globally by both public and private sector customers. Examples of some of the platform's customers include City of Tampere (as Tampere IoT), Danfoss Drives (as DrivePro® Remote Monitoring), Schaeffler (as OPTIME), and Epec (as GlobE). (Wapice, n.d.-a, n.d.-b) With IoT-TICKET, Wapice won the Microsoft Application Innovation Partner of the Year Award from Microsoft in 2019. (Wapice, 2019; Microsoft, 2019)

While IoT-TICKET is cloud native, it is cloud agnostic in that it can be “deployed anywhere” from Microsoft Azure to Amazon Web Services, IBM SoftLayer or in Wapice’s or the customer’s own data centre. The platform is provided to customers either using a PaaS or SaaS service model. (Wapice, n.d.-a)

IoT-TICKET provides out-of-the-box integrations to different software ecosystems and supports connectivity to different devices and systems via protocols such as MQTT and REST. Gateway and edge devices such as Wapice’s own WRM247+ can be used to expand connectivity further to many other protocols such as OPC, OPC-UA, OBD and CAN, although third party hardware is also supported. (Wapice, n.d.-a)

Wapice presents as benefits of the IoT platform the ability to achieve increased efficiency through real-time operational insight, improved productivity and better resource management, cost savings through reduced operating and maintenance costs and the ability to provide better service through process automation, user behaviour prediction and increased responsivity. (Wapice, n.d.-a)

Wapice positions IoT-TICKET as an Internet of Things application and innovation platform. Main components of the platform’s toolset are Dashboards, Cloud Apps, Interface designer, Data-flow editor, Mobile designer, Reports and Report editor as well as Events handling. It is worth noting that terms such as no-code or low-code platform are not used prominently in the platform’s marketing. IoT-TICKET can however be classed both as a low-code and no-code development platform, since while the user is given the option to write traditional JavaScript code as part of their data-flows, this is not required to create solutions with the platform.

Dashboards are web-based applications that an end-user can view and interact with. The user interface of Dashboards is designed using the Interface designer, which in turn utilizes ready-made but configurable widgets. The widgets are divided into four categories: Data-visualization, Input Elements, Miscellaneous, and Layout. When building a Dashboard, the user first selects a widget they want to use by drag-and-dropping it from the widget selection menu onto the design area. After this, they can drag-and-drop onto the widget which datapoint they want to visualize using that widget. The system automatically generates a corresponding data-flow with preset-settings in the Data-flow editor in the background. Figure 4 shows this process in pictures.

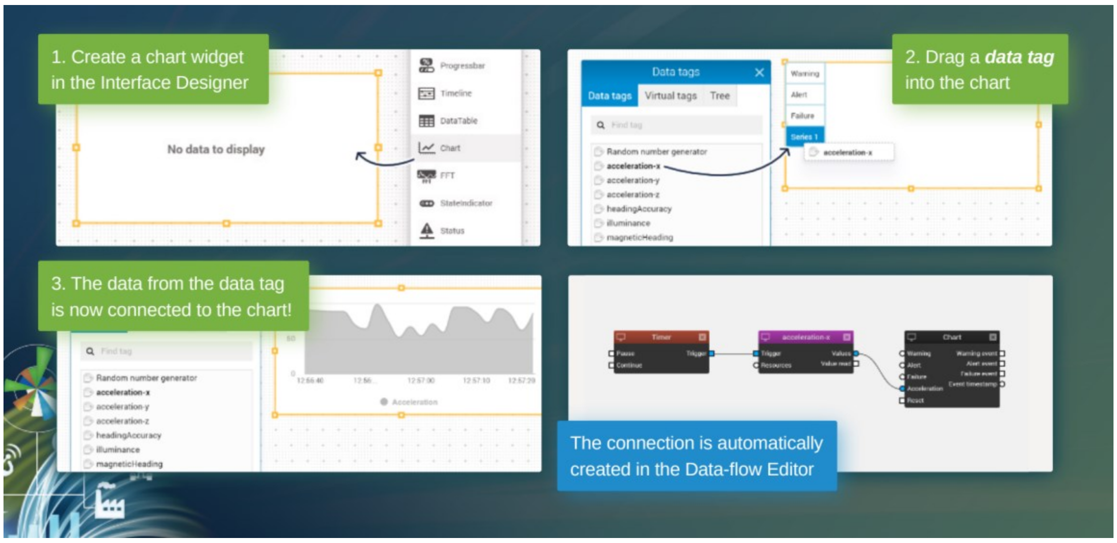
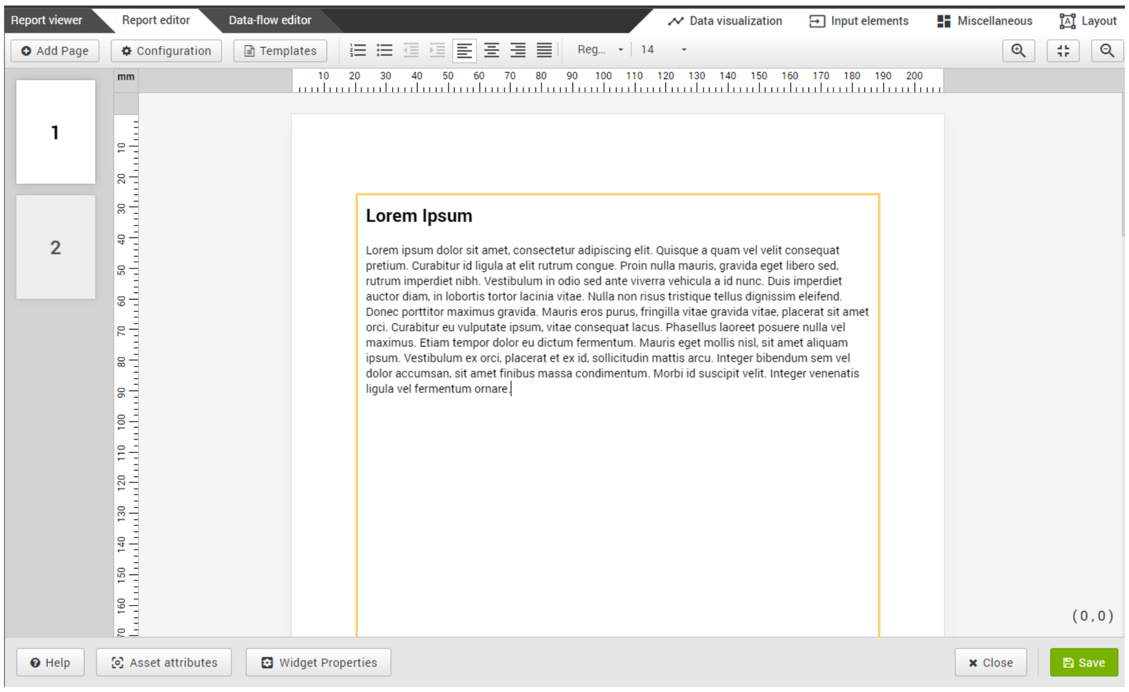

1. Create a chart widget in the Interface Designer
No data to display
2. Drag a *data tag* into the chart
Data tags
Virtual tags
Tree
3. The data from the data tag is now connected to the chart!
The connection is automatically created in the Data-flow Editor
Timer
acceleration-x
Chart

Figure 4 Illustration of the simplified Dashboard design process in Interface designer (Wapice, n.d.-a)

Reports are an alternative way to display data to end-users. Instead of interactive web-applications, reports are .PDF files generated either on a set schedule or as triggered by an event and then sent to the configured users via the set delivery channels, such as via email. The reports are created using the Report editor, which functions similarly to the Interface designer using Widgets and drag-and-drop principles. The report editor interface is shown on Figure 5. Reports can also include Data-flows created using the Data-flow editor.



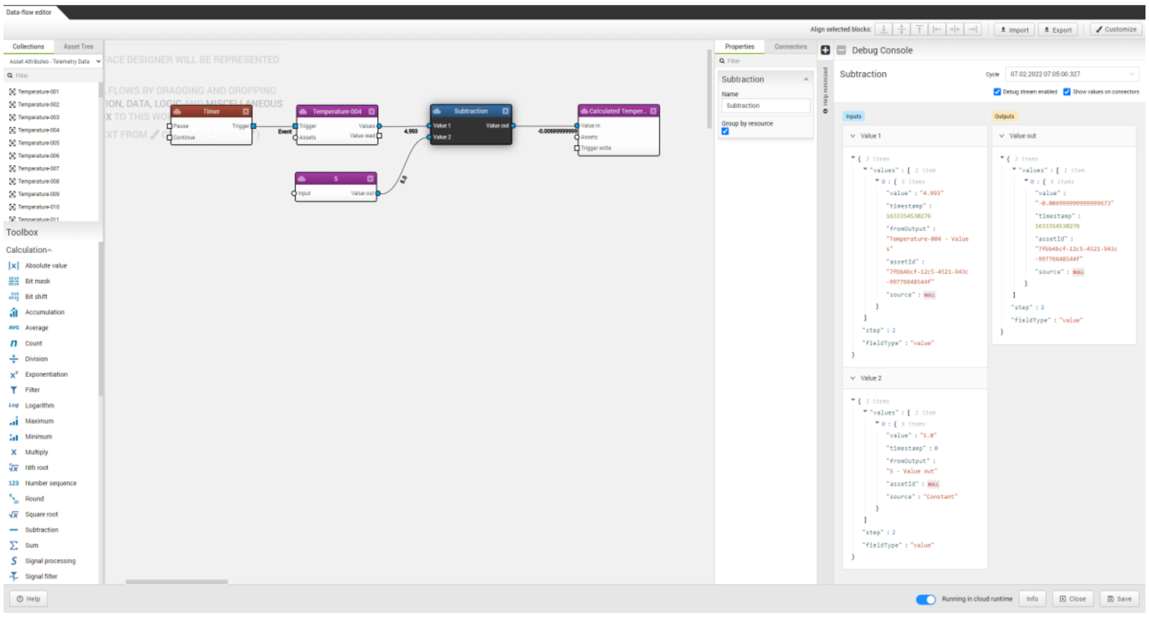
Lorem Ipsum

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quisque a quam vel velit consequat pretium. Curabitur id ligula at elit rutrum congue. Proin nulla mauris, gravida eget libero sed, rutrum imperdiet nibh. Vestibulum in odio sed ante viverra vehicula a id nunc. Duis imperdiet auctor diam, in lobortis tortor lacinia vitae. Nulla non risus tristique tellus dignissim eleifend. Donec porttitor maximus gravida. Mauris eros purus, fringilla vitae gravida vitae, placerat sit amet orci. Curabitur eu vulputate ipsum, vitae consequat lacus. Phasellus laoreet posuere nulla vel maximus. Etiam tempor dolor eu dictum fermentum. Mauris eget mollis nisl, sit amet aliquam ipsum. Vestibulum ex orci, placerat et ex id, sollicitudin mattis arcu. Integer bibendum sem vel dolor accumsan, sit amet finibus massa condimentum. Morbi id suscipit velit. Integer venenatis ligula vel fermentum ornare.

Figure 5 The Report editor

Some applications might not have any visual components to them, but instead only run in the background to complete calculations or logic on the incoming data. These applications are called "Cloud Apps" in IoT-TICKET. Typical use case for them is, for example, to calculate KPIs or to monitor changes in values and thresholds to trigger alarms.

The Data-flow editor is a tool for configuring and visually programming the logic and functionality of the Dashboards, Cloud Apps and Reports. Much like the Interface designer uses ready-made widgets, the Data-flow editor uses blocks that the user can drag-and-drop onto the workspace. The blocks typically have both inputs and outputs that can be used to connect them together by clicking and dragging from one input or output to another. In Data-flow editor the blocks are divided into four categories that are calculation, data, miscellaneous and logic. The user can also, with some limitations, use the script-block to embed into the data-flow code that they have themselves written. Figure 6 shows the Data-flow editor.



Subtraction

Cycle: 07.02.2022 07:05:00.327

Debug stream enabled: ✓
Show values on connectors: ✓

Value 1

```
{ 3 items
  "values" : [ 1 item
    0 : { 5 items
      "value" : "4.993"
      "timestamp" :
      1633354530276
      "fromOutput" :
      "Temperature-004 - Value
      s"
      "assetId" :
      "7fbb4bcf-12c5-4521-943c
      -99776648544f"
      "source" : NULL
    }
  ]
  "step" : 2
  "fieldType" : "value"
}
```

Value 2

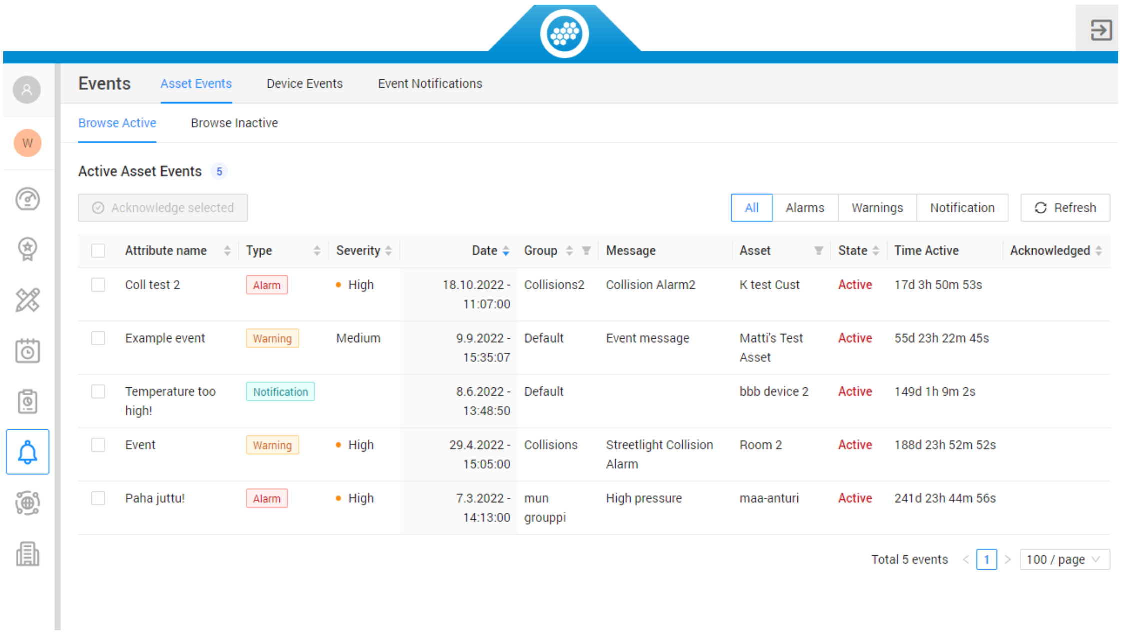
```
{ 3 items
  "values" : [ 1 item
    0 : { 5 items
      "value" : "5.0"
      "timestamp" : 0
      "fromOutput" :
      "5 - Value out"
      "assetId" : NULL
      "source" : "Constant"
    }
  ]
  "step" : 2
  "fieldType" : "value"
}
```

Value out

```
{ 3 items
  "values" : [ 1 item
    0 : { 4 items
      "value" :
      "-0.006999999999999673"
      "timestamp" :
      1633354530276
      "assetId" :
      "7fbb4bcf-12c5-4521-943c
      -99776648544f"
      "source" : NULL
    }
  ]
  "step" : 2
  "fieldType" : "value"
}
```

Figure 6 The Data-flow editor with debugger open on one side.

IoT-TICKET has a built-in Events centre where events triggered in the data-flows or by a device directly can be viewed and acknowledged with a comment. Events can also trigger messages to be sent to configured personnel via email, push notification or other means. Events view is displayed in Figure 7.



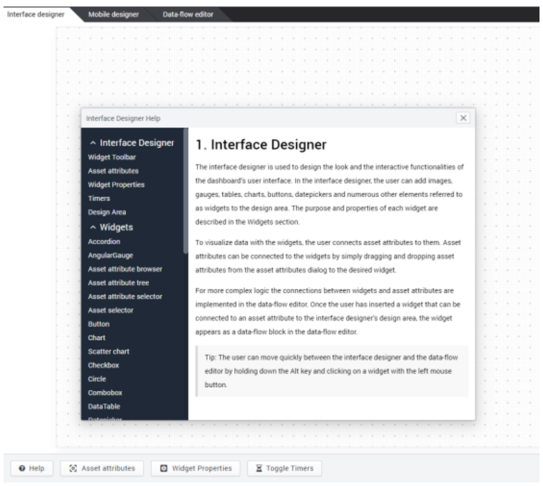
Active Asset Events 5

	Attribute name	Type	Severity	Date	Group	Message	Asset	State	Time Active	Acknowledged
	Coll test 2	Alarm	● High	18.10.2022 - 11:07:00	Collisions2	Collision Alarm2	K test Cust	Active	17d 3h 50m 53s	
	Example event	Warning	Medium	9.9.2022 - 15:35:07	Default	Event message	Matti's Test Asset	Active	55d 23h 22m 45s	
	Temperature too high!	Notification		8.6.2022 - 13:48:50	Default		bbb device 2	Active	149d 1h 9m 2s	
	Event	Warning	● High	29.4.2022 - 15:05:00	Collisions	Streetlight Collision Alarm	Room 2	Active	188d 23h 52m 52s	
	Paha juttu!	Alarm	● High	7.3.2022 - 14:13:00	mun grouppi	High pressure	maa-anturi	Active	241d 23h 44m 56s	

Total 5 events 1 100 / page

Figure 7 The Events view in IoT-TICKET

IoT-TICKET has a built-in documentation that can be accessed in the Interface designer and Data-flow editor. In both views, it can be accessed from the Help-button in the bottom left corner. Additionally, in Data-flow editor the Documentation can be opened as a sidebar that automatically provides information about the data-flow block the user currently has selected. Along with in-person and online training sessions, this serves as the main always up-to-date guide to users. The two documentation views are shown in Figures 8 and 9.



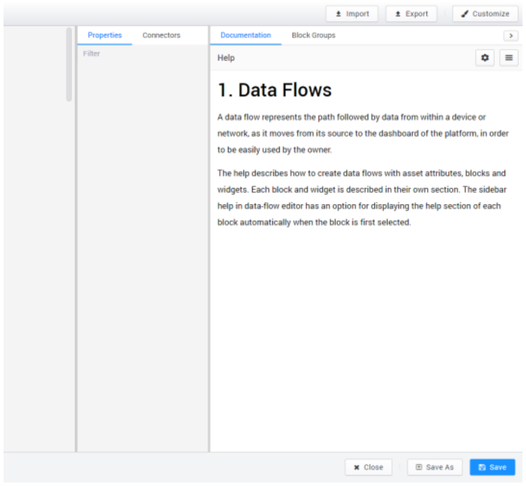
1. Interface Designer

The interface designer is used to design the look and the interactive functionalities of the dashboard's user interface. In the interface designer, the user can add images, gauges, tables, charts, buttons, datepickers and numerous other elements referred to as widgets to the design area. The purpose and properties of each widget are described in the Widgets section.

To visualize data with the widgets, the user connects asset attributes to them. Asset attributes can be connected to the widgets by simply dragging and dropping asset attributes from the asset attributes dialog to the desired widget.

For more complex logic the connections between widgets and asset attributes are implemented in the data-flow editor. Once the user has inserted a widget that can be connected to an asset attribute to the interface designer's design area, the widget appears as a data-flow block in the data-flow editor.

Tip: The user can move quickly between the interface designer and the data-flow editor by holding down the Alt key and clicking on a widget with the left mouse button.



1. Data Flows

A data flow represents the path followed by data from within a device or network, as it moves from its source to the dashboard of the platform, in order to be easily used by the owner.

The help describes how to create data flows with asset attributes, blocks and widgets. Each block and widget is described in their own section. The sidebar help in data-flow editor has an option for displaying the help section of each block automatically when the block is first selected.

Figure 8 (left) Documentation in the Interface designer
Figure 9 (right) Reactive documentation sidebar in the Data-flow editor

3 METHODS

Research methods can largely be divided into qualitative and quantitative methods, which both have their advantages and disadvantages. Quantitative methods fit particularly well into studying natural phenomena objectively with numbers, whereas qualitative methods provide a holistic way to observe more complex social and human phenomena that are better described with words than numbers. Examples of qualitative data collection include fieldwork observation, interviews, and questionnaires. (Jabar et al., 2009)

For this thesis, a qualitative approach was chosen as it allows for an exploratory, descriptive, and emergent way to discover answers to each of the posed research questions. Both qualitative and quantitative methods have been used in previous research to study DX. An advantage of a quantitative approach would have been that it can allow for a larger sample size, at the cost of the lesser descriptiveness of the data. Initially, a multi-method approach of combining both qualitative and quantitative approaches was considered, but it was ultimately deemed to be unsuitable in the face of resources available for conducting the study.

Since the research holds at its base a set of hypotheses that are being tested it can be considered to take a positivist approach. Positivist research is defined by hypotheses testing, measuring of quantifiable variables, and drawing of inferences about phenomena from the sample to a stated population. (Orlikowski & Baroudi, 1991)

Interviews are a common method of conducting qualitative research. They can be typed ranging from structured to unstructured based on the completeness of their script and its restriction to the interviewer's ability to improvise. (Myers & Newman, 2007) Semi-structured interviews present a middle ground between structured interviews where no improvisation is possible at all, and unstructured interviews where there are no pre-prepared questions at all. While not without their downsides, semi-structured interviews offer some of the benefits of both types of interviews by producing comparable data while leaving the door open for the deeper exploration of the answers. For these reasons, semi-structured interviews were selected for this study.

Jacob and Ferguson (2015) instruct that it is important to use existing research literature to guide the questions one prepares for an interview. Doing this, the researcher can develop questions that are both grounded in the existing literature, yet that still have not been answered by it. This study uses at its base the conceptual framework of developer experience originally introduced by Fagerholm & Münch (2012), which has since been expanded by Fagerholm (2015), Nylund (2020), and Lee and Pan (2021). While previous literature on the DX of LCDPs specifically to draw from is scarce, Gao (2022), who measured with a quantitative questionnaire the episodic experience of developers after they had used an LCDP, provided useful metrics for measuring and describing each aspect of the DX framework in this context. Rather than trying to measure objectively the DX of the case platform IoT-TICKET in particular, this study focuses on understanding how the experience of different aspects of the DX differs between two user groups using it. With this in mind, Gao's (2022) metrics were adjusted to suit a more open ended semi-structured interview format. Using this approach, not only can the participants be elicited to give their opinion on if the platform fulfils each metric for DX, but why it does or doesn't do so. From this, we can draw inferences about the different needs and wishes of these two groups, which in turn can be used to inform both future research as well as feature development of the IoT-TICKET platform. Gao (2022) noted that the consideration of the professional developers' backgrounds should specifically include their experience in front-end development, as this is closer to the workflow LCDPs use.

Myers and Nyman (2006) present, based on Erving Goffman's theory of face-to-face interaction, a dramaturgical model for conducting qualitative interviews in Information Systems (IS) research. In this model both parties in the interview are seen as actors that are playing their corresponding parts following a script and working towards a performance. The interview has a beginning and an end, much like a drama has an entry and an exit, and the quality of disclosure is seen as the performance. The interaction between the actors is influenced by not only the physical setting of the interview, but also the organizational, cultural, and social context. They highlight the importance of managing impressions and the interviewer's role as giving 'stage directions' while not over-directing. (Myers & Nyman, 2006)

Further, Myers and Nyman (2006) present seven guidelines for using qualitative interviews, which this study also does its best to adhere to. The guidelines presented are as follows:

1. Situating the researcher as an actor
2. Minimizing social dissonance
3. Representing various “voices”
4. Acknowledging everyone is an interpreter
5. Using mirroring in questions and answers
6. Being flexible
7. Adhering to ethics of interviewing
    a. Permissions
    b. Respect
    c. Fulfilling commitments (confidentiality & results)

As per Myers and Nyman’s (2006) guidelines, and as instructed by Jacob and Ferguson (2015) also, the interviews for this study were prepared for by writing a script that included an opening, introduction, key questions, and closure.

Interviews started with a situation of the researcher as a former employee and current contractor of the commissioning company, and as a person who is doing the research as part of their master’s thesis. To minimize social dissonance between the interviewer and interviewee, a bit of the interviewer’s wider history with the IoT-TICKET platform and background of their current role as a training specialist was described. This was followed by informing the interviewee of confidentiality and ethical guidelines of the research as well as their right to pull out at any point. Following this, demographic and background information was collected and from there the rest of interview itself was smoothly transitioned into.

The parts of the conversations that focused on some of the more negative experiences a user had faced were expanded on by using a laddering interview technique, introduced by Reynolds and Gutman (1988). The laddering interview technique is based on means-end theory with an aim to uncover hierarchies arising from attributes to consequences and to ultimately discover the underlying values relating to them. The use of the laddering technique in this study is supplementary.

Interviewees for the study were provided by Wapice Ltd., who is also the commissioner of this thesis. Potential interview candidates were selected from the company’s IoT-TICKET LCDPs customers per the recommendations of Wapice project managers. While everyone was interviewed individually, based on some prior knowledge about the potential interviewees’ backgrounds the interviewees were split into two groups: the citizen developer group and the professional developer group. This was done in order to receive a more even spread of participants in both groups and to ensure both voices would get heard. To avoid any single IoT-TICKET customer company’s influence in the results being disproportionate, no more than two participants were interviewed from a single customer company.

The interviewees were approached by the author using his company email. In total 10 interview candidates were approached, and a total of 9 interviews were conducted. The interviews were held over Microsoft Teams and they lasted between 30 and 75 minutes. All interviews were recorded to aid in transcribing and the analysis of results, but all recordings were agreed to be kept private and confidential. The interview language varied between Finnish and English depending on the participant’s preference. Interviews were conducted remotely due to the large geographical spread of the participants and the general normalization of remote meetings in the wake of the covid-19 pandemic.

The interviews were coded manually by the author based on the interview recordings and transcripts, with key findings listed and systematically compared using Microsoft Excel. Relevant quotations were highlighted in the transcripts to use in the thesis as examples.

4 RESULTS

The interviews started with the collection of the participant's demographic information. Participants were all between the ages of 25 and 43, with the average age being 34,4 years and the median age being 35 years. Seven of the nine participants were Finnish, while two were German.

Five of the participants had prior professional programming experience while four were citizen developers. Of the five professional programmers, three had experience with front-end development. Participants' experience with IoT-TICKET varied from a minimum of approximately 40 hours to users with over 2000 hours under their belt. Two of the participants described having experience with LCDPs other than IoT-TICKET, but multiple others also brought up similarities between low-code development and PLC programming. Participants demographic details are summarized in the table below. Particular attention was given to the prior front-end development experience of the participants, as suggested by Gao (2022). Demographic details are summarized in Table 3.

Table 3 Demographic information of the participants

Identifier	Age	Prior professional programming experience	Prior LCD experience	IoT-TICKET use time
P1	30	None	None	100-200 hours
P2	43	Yes, including front-end	None	> 2000 hours
P3	33	None	None	> 1000 hours
P4	38	Yes, including front-end	Power BI, Tableau	100-200 hours
P5	35	None	None	100-200 hours
P6	25	Yes, but no front-end	None	~ 60 hours
P7	37	None	None	100-200 hours
P8	25	Yes, including front-end	None	~ 40 hours
P9	44	Yes, but no front-end	IEC 611313	~ 800 hours

The main part of the interviews began with the interviewer requesting the participant to describe what types of solutions they have created using the platform.

Most typical use case for the interviewed users was visualization of IoT data (all of the participants), but significant portion of the participants had also configured and triggered alerts based on tracked values (seven of the nine participants). Just over half (five of the nine participants) described doing some form of further back-end logic or calculations on the data, but no participant stated that they were doing any in-depth analytics. Two of the participants described doing remote control of the connected device in some form. One participant described using the platform to create dashboards for inputting data.

After gaining a rudimentary understanding of the participants' use case for the platform, the first topic to be discussed was how the users saw the platform in terms of helping them achieve their goals. This focuses on the conation aspect of DX. All but one of the participants stated that they were generally able to achieve what they set out to do using the platform, even if this sometimes meant getting support from the platform provider.

The challenges users faced during their use of the platform can be generally divided into three categories: platform feature related (4), user knowledge related (4), and hardware and integration related (1). The two most common strategies the participants used for overcoming these challenges were requesting support from the platform provider (seven participants), as well as persistence and/or trial and error (three participants). Some of the participants used a combination of these strategies. Additionally, two of the participants mentioned that they had amended their initial designs to better fit the platform's features. There did not seem to be a consistent theme as to any roadblocks being more prevalent in either of the groups of users.

Staying on the conation aspect of DX, the participants were requested to describe if they felt using the platform felt either smooth, rough or something in-between. Users in the citizen developer group tended to share the sentiment that the use was smooth (two out of four) or mostly smooth (two out of four). One of the citizen developer users did not answer the question directly, but from the conversation a "mostly smooth" was inferred. The participants from the professional developer group tended to hold more varied stances and apart from one stated that the experience was mixed - some aspects of the platform's use being smooth while some more rough. Only one of the interviewed professional developers described the use as rough. The different aspects of the platform causing this split are explored more in-depth later in this chapter.

Moving onto cognition aspects of the experiencing of DX, all four of the citizen developers described the platform as easy to get started with. However, only two of the five traditional backgrounded developers shared this sentiment initially, the other three noting that it took some time getting used to. Two of the citizen developers described the sense of easiness of use of the platform as having stayed the same over time, whereas two mentioned it having started to feel more difficult as their use cases became more complex than they were in the beginning. This was contrasted by the professional developers, who for the most part considered the system getting easier to use as they used it more and got more accustomed to it.

The platform was described as intuitive to use by most of the participants in both groups. Key feature contributing towards this intuitiveness described by the participants was the drag-and-drop nature of the programming. Participant 7 described their experience with the following.

> "It was simple to find the blocks you needed, and by knowing just some basic principles about programming, you could connect them and quite quickly, bit by bit, build the functionality that you wanted, and then expand on it… Maybe because there's none of that threshold of having to press a lot of keys in just the right order to get code to work, I never got stuck." - Participant 7

Participant 3 noted that the Data-flow editor contributes to its intuitiveness by UI features such as separating the function blocks with different colours based on their type, and by lighting up the inputs that can take the output of a selected connector, as well as refusing to accept incompatible connections.

Apparent simplicity can be a double-edged sword however, as it can lead the user at first into thinking more complex solutions are not possible.

> "If we think about the PLC world, if you have a string connecting blocks, what's transmitted in the connection will be a value - it could be decimal or integer or something - but in IoT-TICKET […] it's a lot more like a digital association that can throughput all sorts of things. […] It can be resources, timestamps, all sorts of things. That was perhaps something that was difficult to learn at first. For a long time, I didn't realize what all it made possible. It looked simplistic, so you thought it was simplistic." - Participant 2

Participants in both groups reported being able to focus on their work while using the LCDP and that the platform, aside from few exceptions, did not cause interruptions in their work. Nearly all of the participants reported having been able to achieve flow while using the platform, with the only user disagreeing stating they had not used the platform continuously long enough to have had the opportunity for it.

Evaluating their feeling of productivity with using the platform, all participants stated that working in low-code made them feel productive or that at worst it had no impact on their productivity.

When prompted to talk about if the platform supported the users' creativity in building solutions with it, the answers had again some spread. Two of the professional developer users remarked that creativity was not a factor in their solution design, two remarked that their creativity was limited but in a way that was positive. One participant remarked that the limiting factor was their knowledge rather than the platform's features. Three of the citizen developers stated that their creativity was not limited, while one agreed with the professional developers who stated that their creativity was limited, but in a positive way.

There were differing opinions amongst the participants on what format of support material is preferred. The current use of integrated documentation in the Interface designer and Data-flow editors received almost universal praise from both groups (8 out of 9 participants). Participant 5, the only participant disagreeing, elaborated on their position by describing it as unhelpful because while it described how a function worked, it didn't tell them what to use it for. Instead, they hoped that the widgets and blocks documentations would provide concrete and easy to understand examples of it in use, instead of (or in addition to) just description of the blocks' operation. This sentiment was shared by three other participants, whom while finding the existing documentation useful also called for a separate tutorial-like support materials in addition to the existing descriptive ones.

While one of the interviewed users called for a booklet-style PDF manual, the idea of a knowledge base -type of external site displaying available information at a single source was universally welcomed by both user groups. The onboarding process was praised and highlighted as important by some users, but others stated they have been happy with the learnability of the system even without one. Videos ranging from "15-30 minutes get-started bootcamp" for the users who have joined the project after onboarding to ones about utilizing newly released features were also commonly requested by the participants in both groups.

Other notable ideas to help with the learnability of the platform brought up by the interviewees were an overlay UI that would guide the user through what each button and part of the system does, as well as a configuration wizard type of UI that would, based on the user's input, generate a template of a Dashboard that the user could start customizing to their needs rather than having to start from a blank canvas.

One of the citizen developer users stated that while they knew a thing was doable with the platform, they might contact support straight away as this was the more efficient way compared to trying to create the solution on their own. Whilst that is an extreme example, users in both groups highlighted their use of the platform provider's support as a mean they would rely on in overcoming roadblocks. The platform provider support was also relied on when it comes to new feature development for specific customer's needs.

As glossed over earlier in discussing the smoothness versus roughness aspect of the LCDP's DX, participants in the professional developer group tended to state their experience of the platform's DX as "mixed" significantly more than their citizen developer counterparts. Three out of the five professional developer users stated that they found the visualization building easy or very easy, while the back-end they would've preferred to develop via traditional programming means. On the other hand, one of the professional developers stated that they would've preferred to do visualizations using traditional programming, but that they appreciated the platform's background functionalities such as data-, device-, and user management for certain kinds of projects. The final remaining professional developer simply stated that it would not even be possible for them to provide the service they provide by hand-coding it and not using an LCDP of some sort. To them, the aspects of LCDP use that they couldn't live without were the efficiency in creating site-specific dashboards and the back-end taking care of data management and connectivity with minimal configuration.

While the amount of data to draw from is limited, it is worth noting that there seems to be a correlation between the user's front-end programming orientation and how they considered the case platform's front-end tools. If we take for example a professional developer without front-end focus, the experience content is largely positive on the front-end features of the platform but less so on back-end development. For professional developers with prior front-end development focus and expertise, the reverse is the case. One possible explanation for this could be that the platform is providing developers with a specific focus the ability to complement their 'weakness' by letting the platform take care of it for them, while perhaps limiting them in the part they do have special expertise in.

Put into other words, looking at this finding through Fagerholm's (2015) model of different aspects of DX and observing as the *experience object* the Interface designer and Data-flow editor respectively, considering which group the user belonged to and if the user had significant prior focus in front-end development as *experience influencer*, it seems that the *experience content* formed through using the platform is distinctly different. Other possible experience influencers could be which version of the system the user is using and if they had attended the onboarding trainings or been provided with access to video tutorials or webinars that help them with using the system. Because of the limited sample size of this study, these potential influencers are unable to be explored here.

Two most commonly listed reason for why the DX sometimes felt abrasive by the users in both groups were repetition and difficulty of keeping track of the big picture in highly complex data-flows. Participant 4, quite eloquently, proposed a "pick two" triangle to describe situations where LCDP use can in their opinion provide an edge over traditional programming. The points on the triangle were the visual design of the dashboard (ie. how good it looks), ease of use & maintainability (ie. how easy it is to navigate and edit), and visualizing complex metrics (ie. how complex the data being visualized is).

> "As long as you only take two of these, it is easy to use and you can get stuff done quickly. As you try to get all three, the data-flow grows massively and you lose the low-code advantage. As it grows past a certain point, it becomes more a burden." – Participant 4

Many of the professional developer users brought up and praised the state of modern DevOps and QA tools available to them when they are programming traditionally. While IoT-TICKET provides users with some tools for debugging and version control, these are limited in comparison. *"Version control being linear is good… but not great"*, Participant 9 summarized, while Participant 8 remarked that *"It's not Git"*.

Most of the users in the professional developer group had used the dashboard templating feature, which is the platform's way of generalizing a dashboard such that it can be instantiated and deployed to different sites or assets, limiting the amount of repetition and ensuring easier maintenance. However, branching new versions of an existing template could be easier according to Participant 1.

Prompted to give ideas about how the issue of data-flow's growing unwieldy could be alleviated, a common request (from three of the professional developers and one citizen developer) was to make data-flow block grouping more powerful. If the block groups could be minimized with only their key information showing, and copied as a whole, this would positively influence both the data-flow legibility and the amount of repetition required with certain types of data-flows. In an even more powerful implementation, it could even make sense for these block groups to be saveable and possible to be reused across multiple Dashboards or Cloud Apps.

On the front-end development side feature requests brought up by the interviewees were an ability to template sections of a dashboard instead of only a full one, have a built-in library for images and icons, and an improved layering functionality that took more inspiration from those of image editing programs with the ability to sort, toggle and lock layers.

Combined for both Interface designer and Data-flow editor, Participant 3 presented a wish for giving the user an option to customize the interface to their liking and specific use case by including a menu, list or view for the user's favourite widgets and blocks á la "My tools".

While being generally very positive about the platform being entirely browser based, Participant 2 brought up as a downside the ease at which one can lose progress by accidentally navigating to another website or by closing the tab they were working on. This has already been addressed in a later version of IoT-TICKET, where the web browser asks user via a pop-up to confirm they really want to exit whenever trying to close or otherwise navigate out of a tab with unsaved changes.

Another source of occasional lost progress is through system instability when under particularly heavy use. A potential mitigation for this aspect of negative DX could be the automated saving of unpublished draft versions of the dashboard being worked on.

While these scenarios have not come to pass, a discussion with Participant 2 led to further reflection on the impact lack of stability of the system could potentially have. *"If the service were offline for eight hours and something critical was to happen [in that time]… that would be very bad."* Participant 2 said. Here, the laddering interview method was used to probe further causes and values behind this remark. The participant noted that extended downtime could result in alerts being not seen and therefore action not being taken to fix the issue active at a site. An issue not being addressed might lead to possible material or even health and safety damages, with the obligation on the service provider to reimburse them. In addition, incidents of downtime can diminish the trust between the service provider and their end-customer. Discussing the importance of this trust, Participant 2 noted that *"if the users' confidence towards the service starts to crumble, it is very difficult to gain back"*, further explaining that in their field it's usually because of lack of trust that the service monitoring subscriptions are cancelled. The participant stated they are content with their level of service and stability of the case platform, but the option for increasing performance by switching to a dedicated instance was also discussed.

Participant 4 discussed his experience with certain bugs he encountered in using the platform while in a project that was under significant time pressure. Again, the laddering interview method was applied to discover more deeply what these issues caused and meant. He stated that some of the bugs made him evaluate his confidence towards the platform and if things would begin to break later, as he kept adding complexity. Especially under the time pressure, these challenges lead to feelings of frustration. Ultimately, he noted that despite the encountered bugs in the designer, once the dashboards were saved things did behave consistently and robustly, and he did find the platform overall reliable.

5 DISCUSSION

This chapter considers and analyses the findings presented in the previous chapter against the research questions and in the context of previous literature. Then, potential issues with the study are addressed.

5.1 Research Question 1

What kind of differences in developer experience of an IoT low-code development platform exist between users with a software developer background and those without?

Research question 1 is closely linked with Hypothesis 1, which asserted that there is a difference in the developer experience of the platform between the two user groups. It seems based on the data that this hypothesis holds true, and certain key differences can be observed. The three main findings are discussed next.

Firstly, citizen developers found the platform easy to use from the get-go more often compared to their software developer backgrounded counterparts. This effect reversed as the users got more familiar with the tool, with the professional developers considering the platform easier to use over time and the citizen developers finding it more difficult.

Kermanchi (2022), who studied both junior and senior software developers as LCDP users, found a significant correlation between the number of years of programming experience and the users preference for traditional versus LCDP programming. In their study, the users with less programming experience felt more in control of and more efficient with LCDPs than the more senior developers, to whom the opposite was true. Regardless of years of programming experience, however, the more familiar users got with LCDPs the more they began considering them efficient. This result seems to be in line with our finding of traditional developers finding the case LCDP easier to use through exposure.

Alyousef (2021) found, contrastingly, that it is easier for traditional developers to use LCDPs than it is for citizen developers. This was attributed to their prior knowledge of traditional software development processes and best practices.

A possible explanation for the difference discovered in this study could be that it takes a moment for professional developers to adjust to the LCDPs way of doing things, but once they have gotten comfortable, they begin to have an advantage over citizen developers thanks to their prior experience.

It is clear that ease of use of LCDPs is not a black and white topic, as is highlighted by Rokis and Kirikova (2022) who found that practitioners in general tend have rather conflicting views: many finding them both easy to study and use, while many also finding them to have a high learning curve. Further study of this aspect is warranted.

Second difference discovered between the citizen developers and professional developers in this study was that citizen developers tended to consider the experience of using the platform smoother than the professional developers did. Based on the interviews this seems to be primarily due to the professional developers looking at the case platform's DevOps tools through the lens of more traditional development environments. An element of this could be that the citizen developers are not as aware of what kinds of supportive processes traditional development entails and therefore don't know to miss them. The aspects causing roughness in the experience are explored more as part of the discussion of RQ2.

Thirdly, citizen developers did not feel their creativity limited by the LCDP as much as their professional developer counterparts did. This limitation, however, wasn't experienced as a negative factor by the professional developers, who tended to state either that creativity wasn't a factor in what they did or that the limitations could even be beneficial.

This is in line with Hallberg (2021), who found the use of LCDPs could be at the same time both inspiring and limiting. Additionally, Dahlberg (2020) found that developers held mixed views on how they think about creativity in LCDPs. On one hand, while traditional development offers the developer many ways to approach a problem, it can at the same time make it difficult to know which approach is the best. LCDPs face a challenge of walking the line between offering limitations that speed up the development of solutions while at the same time not preventing the developers from achieving their goal or hindering their creativity unnecessarily.

5.2 Research Question 2

How can any possibly negative experiences be best addressed?

The primary aspects contributing negatively to DX arose from situations where the user had to either do repetitive tasks, the data-flows of some of the more massive data-flow felt unwieldy, or where the QA and testing tools felt limited. Many of the professional developer user reflected on the state of modern DevOps tools available to them when programming traditionally.

To address these, several improvement ideas, feature requests and possible solutions were gathered and presented to the platform provider. Among these were changes such as improved data-flow block grouping with copying/templating, more advanced debugging and QA tools, improved layering functionality that drew inspiration from image editors, and the ability to customize the designer interface with a "My tools" style menu.

In literature, Khorram et al. (2020) evaluated the testing tools of five prominent LCDPs and proposed a set of feature criteria that can be used to compare the different systems. They highlight the need for high-level automated testing that considers the citizen developer users potential lack of technical knowledge. Due to the WYSIWYG operation nature of case platform, many of the testing approaches recommended may not be suitable to be applied to it. However, this does not mean encouraging testing is not important, and how users are guided to conduct it within the system should be further explored.

IoT-TICKET has recently been updated with new debugging tools for the Cloud App's data-flows, which provides users with much more effective way for debugging their solutions. Some of the interviewed users were still in progress of transitioning to the latest version of the platform and had not yet used these features.

While the platform's built-in documentation received almost universal praise, there were also calls for additional support materials to be made available in a central knowledge base. Besides descriptive documentation, three of the participants called for more tutorial-like material to be created, including videos about how to get started and how to combine blocks and widgets to implement specific features. Ideas about dialog-based configuration wizards to create templates as well as tutorial overlays were also presented.

Al Amin et al. (2021) found that in online discussions about LCDPs, documentation related queries were prominent and difficult to answer, and suggested that platform providers should provide good and effective learning resources to reduce entry level barriers and smoothen the learning curve. Rokis et al. (2023)

also called for addressing the knowledge gap of citizen developers with appropriate training and learning-by-doing.

5.3 Limitations and threats to validity

Like most studies, this study has limitations. Primary limitation to the study's results generalizability is that only the users of a single IoT-focused LCDP were interviewed, meaning it is possible that its results are specific to that LCDP and might not apply to different ones. It is possible, and perhaps to be expected, that different types of LCDPs are experienced differently by their users. The degree to which this is the case presents itself as an opportunity for future research. It may be for example that different types of LCDPs, such as ones that are more form-based rather than ones dealing with timeseries IoT data, could have a significantly different DX in the eyes of their users.

Secondary limitation to this study and its external validity is its limited sample size of nine participants, of which four were citizen developers and five had a professional programmer background. While this number of participants did lead to saturation on the main results, a larger sample could have yielded additional findings.

The impact of any single customer's use case was mitigated by limiting the number of participants from each company to two.

In order to keep the scope of the study manageable, only qualitative approach was used in this study, whereas in an ideal world a quantitative approach could have been employed in conjunction to further validate the findings.

6 CONCLUSION

The purpose of this study was to investigate the differences in developer experience of an IoT low-code development platform between professional developers and citizen developers. Towards this end, qualitative interviews were conducted with nine users, of whom four were citizen developers and five professional developers.

It was found that while there were lots of similarities between the two groups' developer experience with the case platform, there were also differences. Citizen developers tended to initially find the case LCDP easy to use, while the professional developers required a bit of time to adjust to the new way of thinking and the different approach to development. Over time this effect reversed, however, as the professional developers found the case platform easier to use than it had been at first and where citizen developers reported the opposite to have been the case. Secondly, citizen developers tended to find the DX of the platform smoother than their professional developer counterparts, although this could be explained by the professional developers having a point of reference in the modern DevOps tools and processes involved with traditional programming that the citizen developers do not. Main sources of abrasiveness between the platform and its users were occasional repetitive manual tasks and the difficulty of managing the complexity of sometimes very large data-flows. New feature requests to tackle the causes of negative DX as well as ways to assist in the learning of the platform were gathered from the interviews and presented onwards to the platform provider. Finally, citizen developers felt less creatively limited by the LCDP than their professional counterparts, of whom some stated that creativity was not a factor in what they did or that the platform limited their creativity in a positive kind of way.

There is currently a lot of buzz and hype around the topic of LCDPs, but academic literature in this space is still scant, even if quickly expanding. The topic of DX of LCDPs between citizen developers and professional developers was studied here only in the context of a single IoT LCDP, but more attention should be given in the future to how this comparison holds up with both other platforms in the IoT context, as well as entirely different types of LCDPs. Also, the formation and progression of the DX and how this can be influenced by training or provision training materials is highly interesting and worthy of the attention of future studies.

REFERENCES

A Jabar, M., Sidi, F., Selamat, M. H., Abd Ghani, A. A., & Ibrahim, H. (2009). An Investigation into Methods and Concepts of Qualitative Research in Information System Research. *Computer and Information Science*, *2*(4), p47.

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

Alsaadi, H. A., Radain, D. T., Alzahrani, M. M., Alshammari, W. F., Alahmadi, D., & Fakieh, B. (2021). Factors that affect the utilization of low-code development platforms: survey study. *Revista Română de Informatică Și Automatică*, *31*(3), 123–140.

Alyousef, Z. (2021). Challenges Development Teams Face in Low-code Development Process. *Applied Sciences*.

Ashton, K. (2009). That 'internet of things' thing. *RFID Journal*, *22(7)*, 97–114.

Brasil-Silva, R., & Siqueira, F. L. (2022). Metrics to quantify software developer experience: a systematic mapping. *Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing*, 1562–1569.

Bucaioni, A., Cicchetti, A., & Ciccozzi, F. (2022). Modelling in low-code development: a multi-vocal systematic review. *Software and Systems Modeling*, *21*(5), 1959–1981.

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

Cai, F. Z., Huang, S. Y., Kessler, T. S., & Fottner, F. J. (2022). A Case Study: Digitalization of Business Processes of SMEs with Low-Code Method. *IFAC-PapersOnLine*, *55*(10), 1840–1845.

Chin, J., Callaghan, V., & Allouch, S. B. (2019). The Internet-of-Things: Reflections on the past, present and future from a user-centered and smart environment perspective. *Journal of Ambient Intelligence and Smart Environments*, *11*(1), 45–69.

Dahlberg, D. (2020). *Developer Experience of a Low-Code Platform: An exploratory study*. Umeå University.

Di Ruscio, D., Kolovos, D., de Lara, J., Pierantonio, A., Tisi, M., & Wimmer, M. (2022). Low-code development and model-driven engineering: Two sides of the same coin? *Software and Systems Modeling*, *21*(2), 437–446.

Elshan, E., Dickhaut, E., & Ebel, P. (2023). An Investigation of Why Low Code Platforms Provide Answers and New Challenges. *Proceedings of the 56th Hawaii International Conference on System Sciences*.

Fagerholm, F. (2015). *Software Developer Experience: Case Studies in Lean-Agile and Open Source Environments.* University of Helsinki.

Fagerholm, F., & Münch, J. (2012). Developer Experience: Concept and Definition. *2012 International Conference on Software and System Process (ICSSP)*, 73–77.

Gao, D. (2022). *Measuring Developers' Episodic Experience of Low-Code Development Platforms*. Aalto University.

Graziotin, D., Fagerholm, F., Wang, X., & Abrahamsson, P. (2017). 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.

Hallberg, A. (2021). *Using Low-Code Platforms to Collect Patient-Generated Health Data: A Software Developer's Perspective*. Linköping University, Department of Computer and Information Science.

Jacob, S., & Furgerson, S. (2015). Writing Interview Protocols and Conducting Interviews: Tips for Students New to the Field of Qualitative Research. *The Qualitative Report*.

Kermanchi, A. (2022). *Developer Experience in Low-Code Versus Traditional Development Platforms - A Comparative Experiment* [Aalto University].

Khorram, F., Mottu, J.-M., & Sunyé, G. (2020). Challenges & opportunities in low-code testing. *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, 1–10.

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.

Lee, H., & Pan, Y. (2021). Evaluation of the Nomological Validity of Cognitive, Emotional, and Behavioral Factors for the Measurement of Developer Experience. *Applied Sciences*, *11*(17), 7805.

Microsoft. (2019, June 6). *Microsoft announces 2019 Partner of the Year Award winners and finalists*. https://news.microsoft.com/2019/06/06/microsoft-announces-2019-partner-of-the-year-award-winners-and-finalists/

Myers, M. D., & Newman, M. (2007). The qualitative interview in IS research: Examining the craft. *Information and Organization*, *17*(1), 2–26.

Orlikowski, W. J., & Baroudi, J. J. (1991). Studying Information Technology in Organizations: Research Approaches and Assumptions. *Information Systems Research*, *2*(1), 1–28.

Porter, M. E., & Heppelmann, J. E. (2014). How smart, connected products are transforming competition. *Harvard Business Review*, *92.11*, 64–88.

Prinz, N., Rentrop, R., & Huber, M. (2021). Low-Code Development Platforms – A Literature Review. *AMCIS 2021 Proceedings. 2*.

Rafi, S., Akbar, M. A., Sánchez-Gordón, M., & Colomo-Palacios, R. (2022). DevOps Practitioners' Perceptions of the Low-code Trend. *Proceedings of the 16th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 301–306.

Reynolds, T. J., & Gutman, J. (1988). Laddering theory, method, analysis, and interpretation. *Journal of Advertising Research*, 28(1), 11–31.

Richardson, C., & Rymer, J. (2016). *The Forrester Wave™: low-code development platforms, Q2 2016*. Forrester.

Rokis, K., Kirikova, M., & Institute of Applied Computer Systems, Riga Technical University, 6A Kipsalas Street, Riga, LV-1048, Latvia. (2023). Exploring Low-Code Development: A Comprehensive Literature Review. *Complex Systems Informatics and Modeling Quarterly*, *36*, 68–86.

Rokis, K., & Kirikova, M. (2022). Challenges of Low-Code/No-Code Software Development: A Literature Review. In Ē. Nazaruka, K. Sandkuhl, & U. Seigerroth (Eds.), *Perspectives in Business Informatics Research* (Vol. 462, pp. 3–17). Springer International Publishing.

Sahay, A., Indamutsa, A., Di Ruscio, D., & Pierantonio, A. (2020). Supporting the understanding and comparison of low-code development platforms. *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 171–178.

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.

Suresh, P., Daniel, J. V., Parthasarathy, V., & Aswathy, R. H. (2014). A state of the art review on the Internet of Things (IoT) history, technology and fields of deployment. *2014 International Conference on Science Engineering and Management Research (ICSEMR)*, 1–8.

Vincent, P., Iijima, K., Driver, M., Wong, J., & Natis, Y. (2019). Magic quadrant for enterprise low-code application platforms. Gartner Report

Wapice. (2017, June 7). *Digitaalisuuden historia Wapicella*. https://www.wapice.com/fi/insights/digitaalisuuden-historia-wapicella

Wapice. (2019, June 6). *Wapice recognized as Winner for 2019 Microsoft Application Innovation Partner of the Year Award*.

https://www.wapice.com/news/microsoft-application-innovation-partner-of-the-year-award

Wapice. (n.d.a). *IoT-TICKET - Create production-grade IoT applications*. Retrieved August 2, 2023, from https://www.wapice.com/media/brochures/iot-ticket-brochure_en.pdf

Wapice. (n.d.b). *Customers - Wapice*. Retrieved August 2, 2023, from https://www.wapice.com/customers

Waszkowski, R. (2019). Low-code platform for automating business processes in manufacturing. *IFAC-PapersOnLine*, *52*(10), 376–381.

Wortmann, F., & Flüchter, K. (2015). Internet of Things: Technology and Value Added. *Business & Information Systems Engineering*, *57*(3), 221–224.

Yoo, Y., Henfridsson, O., & Lyytinen, K. (2010). The New Organizing Logic of Digital Innovation: An Agenda for Information Systems Research. *Information Systems Research*, *21*(4), 724–735.