Patterns for Low-Code Developer Experience

DANIEL PINHO, Faculty of Engineering, University of Porto and INESC TEC

ADEMAR AGUIAR, Faculty of Engineering, University of Porto and INESC TEC

VASCO AMARAL, NOVA LINCS, DI, FCT/UNL

Low-code development has been empowering end-user developers, who may not have programming or technological backgrounds, to build applications on their own, facing a world with increasing demand for software developers. Developer experience is a concept similar to user experience, but it focuses on the additional role developers play in software development: they create software in addition to using it. There is a need for low-code tool makers to be more aware of the quality of the developer experience they provide, as end-user programmers have increased awareness and expectations of their experience. This paper presents a pattern language to assist the low-code community in improving their developer experience and showcases three patterns that touch upon different dimensions of the mind: Balancing the Scales, Makers' Guidance, and Feasibility Tests.

CCS Concepts: **Software and its engineering** → **Software creation and management** → **Software development process management** → **Software development methods** → **Design patterns** • Human-centered computing → Human computer interaction (HCI) → HCI theory, concepts and models • **Software and its engineering** → **Software notations and tools** → **Development frameworks and environments** → **Integrated and visual development environments**

Additional Key Words and Phrases: developer experience, low-code, patterns, usability

ACM Reference Format:

Pinho, D. et al. 2023. Patterns for Low-Code Developer Experience. HILLSIDE Proc. of Conf. on Pattern Lang. of Prog. 30 (October 2023), 11 pages.

1. INTRODUCTION

Over the past few years, we have seen a rise in the use of **low-code development (LCD)** tools [Rymer and Koplowitz 2019; Vincent et al. 2019], which enable business experts and other professionals without strong programming backgrounds or experience to build applications on their own, taking on the role of **end-user developers (EUDs)**. As their name implies, these tools allow the EUDs to need less code than would be required using traditional programming languages. The rise in the use of LCD tools can be seen as a response to increasing demand for developers since new applications are coming to the market daily while existing services are becoming increasingly complex [Woo 2020].

The concept of **developer experience (DX)** builds upon the idea of **user experience (UX)**. While the latter focuses on what goes on while a person interacts and uses a given tool, the former redirects its attention to the fact that a developer, while using a tool, is also a creator, building something that will be itself used. This added

Corresponding author's address: Daniel Pinho, Faculdade de Engenharia da Universidade do Porto, Rua Dr. Roberto Frias, s/n, 4200-465 Porto, Portugal; email: daniel.pinho [at] fe.up.pt

responsibility brings some differences in the needs of developers, especially when compared to users [Fagerholm and Münch 2012].

The difference in paradigm existing in LCD tools compared to traditional code-based programming brings questions about the experience they provide to the EUDs that use said tools. Expectations, needs, and backgrounds may differ, but the dual user/creator role is maintained.

As such, this work aims to document patterns contributing to a high-quality DX in LCD contexts based on examples from the LCD industry and the literature. The patterns discussed herein are targeted mainly at users of LCD tools; however, some fall within the sphere of influence of platform vendors. The remainder of this paper is as follows: Section 2 expands further on these two concepts while exploring some related work. Section 3 provides an overview of a possible pattern language for LCD-oriented DX and presents three initial patterns. The first one, BALANCING THE SCALES, suggests finding the ideal balance between the different dimensions of the mind to identify which factors have a significant influence on the DX in play. The second, MAKERS' GUIDANCE, advises LCD tool makers to provide informative materials to support skill growth. The third pattern, FEASIBILITY TESTS, recommends employing feasibility tests to assess the usability of the LCD tool by the tool makers. Finally, Section 4 provides a conclusion to this paper.

2. BACKGROUND AND RELATED WORK

This section provides information on the two main concepts of this paper, low-code development and developer experience, while also in the meantime detailing related work.

2.1 Low-code development

As their name suggests, low-code tools enable application development without requiring many lines of code to be written. As a concept, LCD has existed at least since the 1980s, and developers have always tended to move from lower-level programming languages to higher-level ones and to provide more and more layers of abstraction, enabling the developers to think less about at the machine level and more on what they are trying to implement.

While the term "low-code" was only coined by Forrester in 2014 [Richardson et al. 2014], we could already find solutions in the market by then that are considered today to be LCD tools, such as OutSystems and Mendix. Currently, these two vendors, alongside Microsoft Power Apps and Salesforce, are identified by Forrester [Rymer and Koplowitz 2019] and by Gartner [Vincent et al. 2019] as market leaders in LCD solutions.

These tools enable the development of various applications, with business applications, web and mobile apps, user interfaces and database management programs most commonly described in the literature [Pinho et al. 2023].

Despite the growing industry, the LCD concept has been the target of some academic discourse regarding its nature, particularly concerning its relationship with model-driven approaches (MD*). Model-driven development is a methodology that uses models as the primary development artefact, as they represent existing concepts and allow developers to raise abstraction levels. The models are modified using transformations to build software, enabling the automation of several steps [Brambilla et al. 2012]. In a position paper, Cabot [Cabot 2020] argues that LCD is a subset of MD* approaches. In a contrasting opinion, Di Ruscio et al. [Di Ruscio et al. 2022] position these two concepts as intersecting, each with areas where they are not the same (e.g. a low-code tool may not use models in its underlying architecture). Regardless of the nature of this relationship, the common elements between MD* and LCD made it possible for us to look at MD* patterns and transform them into LCD-oriented ones in a previous article [Pinho et al. 2021].

Practitioners in the industry have not agreed on a definition for LCD, with the same sentiment being shared in academic publications. In a previous article, we [Pinho et al. 2023] proposed a definition for this concept as "a set of approaches, technologies, and tools that enable rapid application development through techniques that reduce the amount of code written." In this work, we will be following this definition when discussing LCD.

2.2 Developer Experience

DX was coined by Fagerholm and Münch [Fagerholm and Münch 2012] as a response to the fact that developers have unique needs compared to regular end-users and that the definition of UX did not cover all these different needs due to differing end goals. The DX concept targets everyone that is involved in software development activities.

Fagerholm and Münch included different aspects based on the trilogy of mind theory commonly discussed in psychology [Hilgard 1980]. These aspects are the cognit dimension, related to developers' perception of the infrastructure they interact with; the affective dimension, which deals with how developers feel about their work; and the conative dimension, linked to the value developers give their work [Fagerholm and Münch 2012].

The literature has examples of works ([Kuusinen 2016; Kuusinen et al. 2016; Morales et al. 2019; Nylund 2020] that expand upon relevant concepts, discussing motivation and flow, among other concepts.

3. DX IN LCD PATTERNS

This section describes the process used for pattern mining, the pattern language we're presenting, and showcases three patterns: BALANCING THE SCALES, MAKERS' GUIDANCE, and FEASIBILITY TESTS.

3.1 Pattern mining process

The pattern mining process was conducted taking into consideration two realms: academic publications and industry knowledge.

For the first realm, we considered peer-reviewed publications with low-code development as a primary topic. These were found through searches in search engines such as Google Scholar[1] and Scopus[2], along with articles found in the process of conducting a previous study related to LCD [Pinho et al. 2023] and the results that came from said study.

Regarding industry knowledge, we gathered two types of information: empirical knowledge, based on first-hand use of LCD platforms such as OutSystems and Microsoft Power Apps, and information available on the tool makers' websites, such as documentation.

3.2 Pattern language overview

The patterns described here use the Alexandrian form of name-context-problem-solution, accompanied by a set of forces and one of example applications. Relationships between patterns, whether they are from this pattern language or another one, are described whenever relevant. The pattern language's primary target audience is the users of LCD tools (referred to as End-User Developers, EUDs, in this document); however, some patterns are within the scope of the tool makers. In this document, this information is present in a pattern's full text through a symbol next to its name: patterns that target EUDs are signalled by a four-pointed star (✦), while those targeting tool makers are signalled by a six-pointed star (✶).

The pattern language considers three main types of patterns, *Cognitive*, *Affective*, and *Conative Patterns*, which map themselves to each of the mental dimensions of the same names present in Fagerholm and Münch's definition of DX [Fagerholm and Münch 2012]. These types are not disjoint; a pattern can be associated with more than one simultaneously if they can influence the concept described by the latter. The pattern candidates PERCEIVED INFRASTRUCTURE, GOOD FEELINGS ABOUT WORK, and WELL-VALUED CONTRIBUTION are direct mappings to the types of patterns as mentioned above, respectively. The types of the remaining patterns can be derived from the relationships between them and these three pattern candidates.

Figure 1 provides an overview of the pattern candidates and how they relate to each other. Patterns described in their entirety are signaled with a rectangle outline, while the three patterns that are directly mapped to each

[1] https://scholar.google.com
[2] https://www.scopus.com/

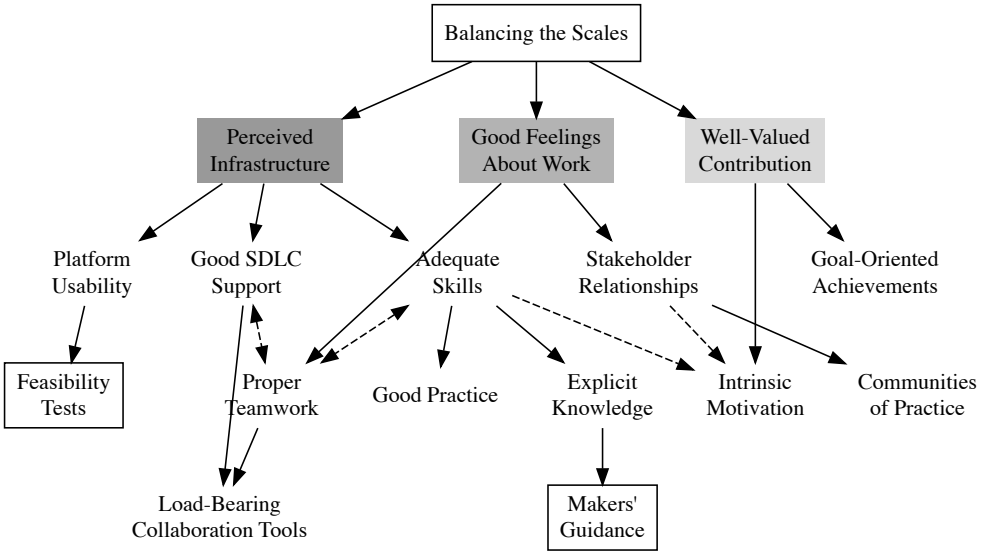


Fig. 1. The candidate patterns for the pattern language. A rectangle outline indicates the patterns that are described in their entirety, while pattern candidates directly mapped to a DX dimension are shaded in gray.

of the DX dimensions (PERCEIVED INFRASTRUCTURE, GOOD FEELINGS ABOUT WORK, and WELL-VALUED CONTRIBUTION) are shaded in gray. Full arrows indicate that the usage of a pattern leads to another one, while dashed arrows indicate patterns that influence other ones.

Table I lists the patterns described in detail in this work, declaring their categories and listing each pattern's patlet. Table II (see Appendix A) provides details on the remaining pattern candidates in the pattern language.

3.3 Pattern: BALANCING THE SCALES ✦

Context. Low-code development (LCD) enables the users of such tools (end-user developers, EUDs) to develop software without relying much on traditional, code-oriented paradigms, technologies and languages. This makes it possible for people without technical backgrounds and programming skills to implement features in a given software project. These lower barriers of entry can be seen in LCD tools marketing themselves as user-friendly, typically regarded as having good usability.

Table I. Pattern overview.

Pattern	Type	Patlet
BALANCING THE SCALES ✦	Cognitive, Affective, Conative	There is always a set of conditions currently influencing a given person's DX. This DX is influenced by three main dimensions, with each one of them having a different weight depending on the person. By identifying their relative importance, one can devise a plan for improvement.
MAKERS' GUIDANCE ✦✶	Cognitive	Provide the end-user developers with various informative materials, such as documentation, guides, tutorials, and classes to support their learning process and increase their skills using a low-code tool.
FEASIBILITY TESTS ✶	Cognitive	Perform usability-oriented tests, such as the NASA TLX or the System Usability Scale, to prevent usability issues in the long run.

Problem. With an increase in perceived user-friendliness and accessibility comes an implicit demand that EUDs will have a positive developer experience (DX) [Pinho et al. 2023]. In reality, LCD tools come with their own challenges, including their learning curves, vendor lock-in, and having to write more code than expected. These challenges affect each person's DX in different ways.

How can we maximise the quality of DX-facing issues?

Forces

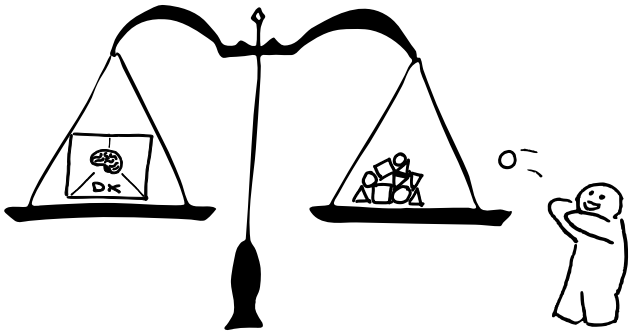
Are we putting a square peg in a round hole?. As every situation is different, a given LCD tool will influence the DX uniquely. Different tools have differing sets of features; it is important to consider if the feature set in play is the most appropriate for the current work. How strong are the EUD's skills to make the most use of the features available? Does the tool have strong usability, or is it too obtuse to use despite having the most appropriate features? These questions affect the EUD's perception of their infrastructure while also influencing the DX an EUD experiences.

How was the tool chosen?. Each situation is different; one may be working on a personal project where choices regarding technology can be made more freely. Another developer may be using a tool at work in a team setting, where this choice was already made for them. This sense of agency can influence the value a person gives to their work and their feelings about it.

How is the team environment?. Work in a LCD setting, in a similar way to traditional, code-based approaches, involves several people of differing roles. While only some may work with the LCD tool in use, the interactions between team members may influence how they feel about their work and how they value their contribution to the project.

Solution

Assess the impact of the LCD tool and the environment where it is used on each one of the dimensions of the mind: cognitive, affective, and conative. Considering the limitations in place, assess the importance of each dimension relative to the other ones and focus more on improving the most important ones.



To better guide one in the improvement of the existing DX, its three different dimensions must be considered, as each has its own intricacies and details.

The **cognitive** dimension deals with the way developers perceive the infrastructure in use. This perception may be influenced by factors such as skill, techniques, and process. PERCEIVED INFRASTRUCTURE delves further into this dimension.

The **affective** dimension is related to how developers feel about their work, taking, among others, social, team, and belonging factors into account. GOOD FEELINGS ABOUT WORK explores this dimension.

The **conative** dimension has to do with how developers see the value of their work. It takes factors such as motivation, commitment and goal-setting into consideration. For further information on this dimension, refer to WELL-VALUED CONTRIBUTION.

The weight of each dimension will vary from person to person, as each developer has different needs and can tolerate different things. Each situation may differ in terms of environments and people involved. Finding the level of influence each of them has on the situation will require some introspection.

Consequences. Performing this assessment can help one identify which factors matter the most to them. In addition to taking improvement steps described in other patterns, this assessment can contribute to a better DX by focusing on what matters most that can also undergo changes.

There is the risk that the factors that influence a person's DX are outside of their sphere of influence; their impact can still be softened by improving factors that can be changed.

One must also consider that this assessment takes time. Knowing what has the most impact on one's DX requires careful reflection, and depending on one's workload, it may not be easy to find time to think about these topics and experiment with the different factors in play.

Related patterns. As this pattern's solution indicates, this pattern leads to PERCEIVED INFRASTRUCTURE, GOOD FEELINGS ABOUT WORK and WELL-VALUED CONTRIBUTION.

3.4 Pattern: MAKERS' GUIDANCE ✦✶

Context. Like any other software, LCD tools have their own features, processes and capabilities. However, unlike most user-oriented software where the user simply interacts with an application, LCD platforms enable EUDs to build applications using various techniques that raise the abstraction level in the process [Benac and Mohd 2021; Bock and Frank 2021]. As such, EUDs play both a user role and a creator role [Fagerholm and Münch 2012], where they have a specific set of needs.

Problem. As LCD tools are used to develop software, similarly to traditional IDEs, they incur some complexity; for instance, a low-code tool must have constraints to prevent illegal operations and should perform checks to verify the validity of the EUD's input. This complexity can influence the learning curve of a given LCD tool; the variety of settings and functions can take some time to get used to them as the user climbs the learning curve.

How can EUDs acquire knowledge on how to use an LCD tool effectively?

Forces

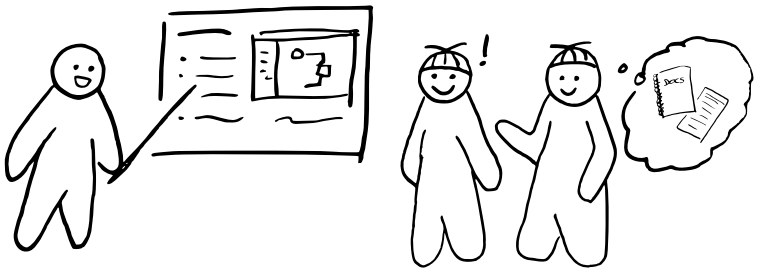
What do the users need?. While some people who use LCD tools have technical backgrounds and have programmed using traditional programming languages in the past, others may be business and domain experts empowered by a low-code tool's apparent ease of use. As such, each user has their own skill-set that exists *a priori*, and what they need to learn at the moment changes from person to person.

What kind of resources do the users have?. Depending on the situation (such as, for example, developing an app in a corporate environment versus as a hobby), a EUD may find themselves with a certain amount of time and budget—the presence (or lack thereof) of these resources influences which learning approaches are viable.

Where is information coming from?. As is also the case with traditional programming languages, the tool makers, along with the respective communities, can be rich sources of information. The tool makers should know the software they programmed, while community members have practical information gathered over time. Differing knowledge sources have different points of view on the tool, and they can be taken into account.

Solution

LCD tool makers should consider making different forms of information available about the tool in question, enabling EUDs to develop their skills.



This documentation should exist in several different forms to adhere to the diverse needs of users. Some may benefit from simple reference pages outlining features and functions similar to the documentation pages for a programming language. In contrast, others may require a getting started guide or a tutorial that introduces different concepts in a structured way. These are ways where EXPLICIT KNOWLEDGE can be found.

Depending on the tool and the resources available to the EUD, they can also consider taking classes or workshops. These enable the EUD to clarify any questions, potentially strengthening the learning process. They also provide a way for EUDs to get their hands busy with the tool, a form of PRACTICE MAKES PERFECT.

These different types of documentation do not necessarily need to be created by the tool makers; several tools have thriving communities, enabling information to be shared using, for example, blog posts, forums, or even videos. Tool makers can facilitate the usefulness of this information by making use of their position as a reference point regarding their tool by sharing these types of information to the community of EUDs that use that tool.

Depending on the size of the community, events may also be a viable way to transmit knowledge; they provide a controlled space for the tool makers and EUDs to connect, while the social aspect can strengthen a person's motivation about using the tool.

Consequences. Having learning materials and making use of them will help EUDs be more familiar and knowledgeable regarind the tool or platform in use, supporting the development of ADEQUATE SKILLS.

On the other hand, managing and maintaining knowledge bases, as well as other learning materials, takes time and effort. If a tool maker wants to provide these materials themselves, they should be updated as features are added or changed to ensure they reflect the current state of the tool.

Known Uses. OutSystems, Mendix, Pega, Appian, and Microsoft have sections on their websites dedicated to learning, where they provide these different types of information [Appian 2023; Mendix 2023; Microsoft 2023; OutSystems 2023; Pega 2023].

Related Patterns. This pattern details one of the ways EUDs can find EXPLICIT KNOWLEDGE about the tools they are using.

3.5 Pattern: FEASIBILITY TESTS ✶

Context. LCD tools are complex programs; they enable EUDs to develop software without writing code for the most part, usually exchanging traditional programming paradigms for a visual, drag-and-drop-based toolset [Di Ruscio et al. 2022; Pinho et al. 2023]. These provide a higher abstraction level, enabling EUDs without extensive technical backgrounds to participate in the software development process in new ways while accelerating that process.

Problem. Visual programming tools are limited so that every element must have a visual representation. A variety of types of entities requires different symbols for each one, which can impair legibility depending on the number of symbols in place. In addition, as the program's complexity increases, the number of nodes and the relationship between them can increase significantly [Nickerson 1994], making the program require more work to understand and work on.

There are other potential usability issues, such as a lack of knowledge about the LCD tool's capabilities or even difficulties in having interoperability between the tool and other programs [Pinho et al. 2023].

How can the Makers be sure that their LCD tool has good usability?

Forces

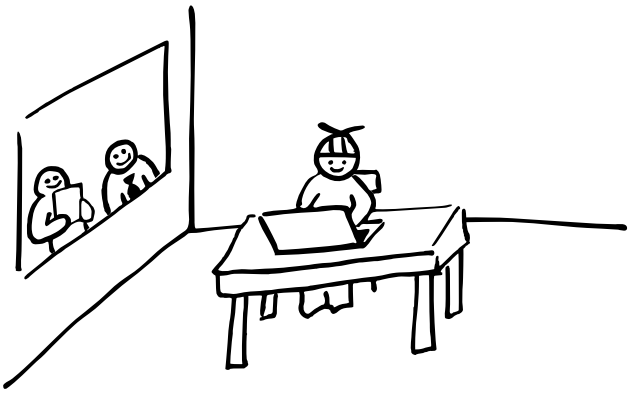
What are the users expecting?. Thanks to LCD tools, EUDs have enjoyed a lower barrier of entry for software development. However, depending on their background, the EUDs may have increased expectations regarding the quality of the usability of the tool they are using, particularly compared to a traditional IDE.

How consistent is the tool's UI?. The UI of a given program is expected to follow a consistent design language. In more extensive LCD solutions, as each component is more distanced from the other, there is the risk that some parts may feel disjointed from each other.

How flexible are the tools?. As LCD tools are used to create applications, they should be flexible enough to enable various types of end results. Any existing limitations can hinder the development process, and if these are unexpected, they can turn a person's motivation and productivity.

Solution

At regular time intervals, conduct feasibility studies to assert that the LCD tool's usability is as expected and act upon the findings.



Depending on the size of the tool, these studies can sometimes target just a specific part; different components may evolve at different paces, and faster-changing ones may require more frequent assessments.

Different types of feasibility studies can be applied, such as the NASA Task Load Index (TLX) [Hart 1986], the System Usability Scale (SUS) [Brooke 1996], and performing heuristic evaluations of the UIs, as described by Nielsen and Molich [Nielsen and Molich 1990]. As the study results fluctuate, the Makers should know how and when to take action regarding specific components of the LCD tool.

Consequences. Conducting feasibility tests comes with a cost of resources such as time and budget before and after application. Before the tests, they should be appropriately devised to ensure relevant information is acquired, and the Makers have to invite their users to participate. The time required to conduct the tests needs to be considered depending on the number of participants. Finally, the results should be analysed, and their evolution over time should be tracked.

On the other hand, a well-designed feasibility test should yield relevant insights regarding how a low-code tool (or a specific component thereof) is used, which can bolster the PLATFORM USABILITY of the tool.

Examples. Sinha et al. [Sinha et al. 2020] developed a DevOps approach accessible to business users, presenting an evaluation of the tool's effectiveness in the same paper.

Henriques et al. [Henriques et al. 2018] proposed a new notation for an OutSystems DSL with usability issues, evaluating both the original and proposed versions with the TLX and SUS methods.

Tamilselvam et al. [Tamilselvam et al. 2019] proposed a low-code IDE targeted at deep learning and performed SUS and TLX studies to assess its usability.

Related patterns. The tests' findings can identify problem areas requiring more information about their usage, which can be provided using MAKERS' GUIDANCE. The relationships developed between the Makers and the EUDs while running the tests are a form of STAKEHOLDER RELATIONSHIPS, which can be beneficial for both sides.

4. CONCLUSION AND FUTURE WORK

Low-code development makes building applications possible without writing many lines of code, lowering the barrier of entry for end-user developers. LCD is growing both in the industry and academia and low-code tool makers must be mindful of the quality of the developer experience EUDs encounter.

In this paper, we looked at the literature and the industry to document patterns that positively impact the DX encountered by EUDs while working with LCD tools. Alongside a set of pattern candidates, the three patterns in this paper discussed in more detail, BALANCING THE SCALES, MAKERS' GUIDANCE, and FEASIBILITY TESTS, touch upon different dimensions in a developer's mind.

In the future, we expect to continue work on this pattern language, documenting more patterns, revising existing documented ones and strengthening the relationships between them.

ACKNOWLEDGMENTS

The authors would like to thank Eduardo Guerra for his assistance and support during the shepherding process. They would also like to thank Guerra, Abayomi Agbeyangi, Steve Berczuk, Kazuki Hioki, Erika Inoue, and Hiroki Itoh for their helpful comments during the Writer's Workshop this paper participated in.

A. PATTERN LANGUAGE PATLETS

This appendix provides patlets for each pattern candidate for the pattern language. This information is present in Table II.

REFERENCES

Appian. 2023. Resource Center - Free White Papers, Case Studies, Analyst Reports | Appian. (2023). https://appian.com/learn/resources/resource-center-all.html Accessed on 2023-05-27.

Ryan Benac and Tauheed Khan Mohd. 2021. Recent Trends in Software Development: Low-Code Solutions. In *Proceedings of the Future Technologies Conference (FTC) 2021, Volume 3 (Lecture Notes in Networks and Systems)*, Kohei Arai (Ed.). Springer International Publishing, 525–533. DOI:http://dx.doi.org/10.1007/978-3-030-89912-7_41

Alexander C. Bock and Ulrich Frank. 2021. Low-Code Platform. 63, 6 (2021), 733–740. DOI:http://dx.doi.org/10.1007/s12599-021-00726-8

Marco Brambilla, Jordi Cabot, and Manuel Wimmer. 2012. *Model-Driven Software Engineering in Practice*. Number 1 in Synthesis Lectures on Software Engineering. Morgan & Claypool.

John Brooke. 1996. SUS – a Quick and Dirty Usability Scale. 189–194.

Jordi Cabot. 2020. Positioning of the Low-Code Movement within the Field of Model-Driven Engineering. In *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings* (2020-10-16) *(MODELS '20)*. Association for Computing Machinery, 1–3. DOI:http://dx.doi.org/10/gh78b6

Davide Di Ruscio, Dimitris Kolovos, Juan de Lara, Alfonso Pierantonio, Massimo Tisi, and Manuel Wimmer. 2022. Low-Code Development and Model-Driven Engineering: Two Sides of the Same Coin? (2022). DOI:http://dx.doi.org/10.1007/s10270-021-00970-2

Fabian Fagerholm and Jurgen Münch. 2012. Developer Experience: Concept and Definition. In *2012 International Conference on Software and System Process (ICSSP)*. IEEE, 73–77. DOI:http://dx.doi.org/10.1109/ICSSP.2012.6225984

Table II. Pattern candidates' patlets.

Pattern	Type	Patlet
BALANCING THE SCALES	Cognitive, Affective, Conative	There is always a set of conditions currently influencing a given person's DX. This DX is influenced by three main dimensions, with each one of them having a different weight depending on the person. By identifying their relative importance, one can devise a plan for improvement.
PERCEIVED INFRASTRUCTURE	Cognitive	A low code tool is a primary element of a EUD's daily activities, along with the procedures and skills necessary. There should be an alignment between these three to remove any barriers to work.
GOOD FEELINGS ABOUT WORK	Affective	To deal with the highly mental nature of low code development, you should have good feelings about your work, the people you work with and the environment around you.
WELL-VALUED CONTRIBUTION	Conative	Through introspection, look into how valuable you feel your contribution is, not only to the project but to yourself and to your own goals.
GOAL-ORIENTED ACHIEVEMENTS	Conative	A low code tool should enable its users to achieve their goals effectively. These goals should be well construed.
INTRINSIC MOTIVATION	Conative	A key element to effective work is the EUD's intrinsic motivation. There should be enough factors that facilitate motivation in play.
STAKEHOLDER RELATIONSHIPS	Affective	LCD involves, just like any other development paradigm, communication between different stakeholders. These relationships should be tended to to remove any possible obstacles.
COMMUNITIES OF PRACTICE	Affective	To facilitate the sharing of knowledge (both explicit and tacit) between parties interested in the same topics, they should organise themselves into communities targeting those topics.
GOOD SDLC SUPPORT	Cognitive	The LCD tools should support the may diverse tasks present in the software development lifecycle, such as requirements engineering, designing, testing, and deployment.
PROPER TEAMWORK	Affective	The LCD tools should support collaboration between developers and stakeholders, minimising any friction points.
ADEQUATE SKILLS	Cognitive	To ensure effective and quick development, the EUDs should develop their skills, enabling them to implement what they intend to.
EXPLICIT KNOWLEDGE	Cognitive	Explicit knowledge, such as discussion forums, documentation, and classes enable EUDs to develop their skills in a structured way.
GOOD PRACTICE	Cognitive	Develop tacit knowledge through practice and experience, enabling the EUD to feel more at ease and be more skillful.

Sandra G. Hart. 1986. NASA Task Load Index (TLX). (1986). https://ntrs.nasa.gov/citations/20000021487

Henrique Henriques, Hugo Lourenço, Vasco Amaral, and Miguel Goulão. 2018. Improving the Developer Experience with a Low-Code Process Modelling Language. In *Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems - MODELS '18*. ACM Press, 200–210. DOI:http://dx.doi.org/10.1145/3239372.3239387

Ernest R. Hilgard. 1980. The Trilogy of Mind: Cognition, Affection, and Conation. 16, 2 (1980), 107–117. DOI:http://dx.doi.org/10.1002/1520-6696(198004)16:2<107::AID-JHBS2300160202>3.0.CO;2-Y

Kati Kuusinen. 2016. Are Software Developers Just Users of Development Tools? Assessing Developer Experience of a Graphical User Interface Designer. In *Human-Centered and Error-Resilient Systems Development*, Cristian Bogdan, Jan Gulliksen, Stefan Sauer, Peter Forbrig, Marco Winckler, Chris Johnson, Philippe Palanque, Regina Bernhaupt, and Filip Kis (Eds.). Vol. 9856. Springer International Publishing, 215–233. DOI:http://dx.doi.org/10.1007/978-3-319-44902-9_14

Kati Kuusinen, Helen Petrie, Fabian Fagerholm, and Tommi Mikkonen. 2016. Flow, Intrinsic Motivation, and Developer Experience in Software Engineering. (2016), 104–117. DOI:http://dx.doi.org/10/ghhdzz

Mendix. 2023. Welcome to Mendix Docs. (2023). https://docs.mendix.com/ Accessed on 2023-05-27.

Microsoft. 2023. Microsoft Power Apps Documentation - Power Apps. (2023). https://learn.microsoft.com/en-us/power-apps/ Accessed on 2023-05-27.

J. Morales, C. Rusu, F. Botella, and D. Quiñones. 2019. Programmer eXperience: A Systematic Literature Review. 7 (2019), 71079–71094. DOI:http://dx.doi.org/10/ghp39h

J.V. Nickerson. 1994. Visual Programming: Limits of Graphic Representation. In *Proceedings of 1994 IEEE Symposium on Visual Languages* (1994-10). 178–179. DOI:http://dx.doi.org/10.1109/VL.1994.363624

Jakob Nielsen and Rolf Molich. 1990. Heuristic Evaluation of User Interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (1990-03-01) *(CHI '90)*. Association for Computing Machinery, 249–256. DOI:http://dx.doi.org/10.1145/97243.97281

Anders Nylund. 2020. A Multivocal Literature Review on Developer Experience. (2020). http://urn.fi/URN:NBN:fi:aalto-202003222600

OutSystems. 2023. OutSystems Training: Learn to Develop Mobile and Web Apps. (2023). https://www.outsystems.com/training/ Accessed on 2023-05-27.

Pega. 2023. Pega Academy Home | Pega Academy. (2023). https://academy.pega.com/ Accessed on 2023-05-27.

Daniel Pinho, Ademar Aguiar, and Vasco Amaral. 2021. Mining Good Practices of Low-Code Software Development from Model-Driven Approaches. In *Proceedings of the 28th Conference on Pattern Languages of Programs (to be published) (PLoP '21)*. Association for Computing Machinery.

Daniel Pinho, Ademar Aguiar, and Vasco Amaral. 2023. What about the Usability in Low-Code Platforms? A Systematic Literature Review. 74 (2023), 101185. DOI:http://dx.doi.org/10.1016/j.cola.2022.101185

Clay Richardson, John R Rymer, Christopher Mines, Alex Cullen, and Dominique Whittaker. 2014. New Development Platforms Emerge for Customer-Facing Applications. (2014). https://www.forrester.com/report/New-Development-Platforms-Emerge-For-CustomerFacing-Applications/RES113411

John R Rymer and Rob Koplowitz. 2019. *The Forrester Wave™: Low-code Development Platforms for AD&D Frofessionals, Q1 2019*. Forrester Research.

S. Sinha, T. Astigarraga, R.B. Hull, N. Jean-Louis, V. Sreedhar, H. Chen, L.X. Hu, F.E. Carpi, J.A.B. Cannata, and W. Loach. 2020. Auto-Generation of Domain-Specific Systems: Cloud-hosted Devops for Business *(IEEE International Conference on Cloud Computing, CLOUD)*, Vol. 2020-October. 219–228. DOI:http://dx.doi.org/10.1109/CLOUD49709.2020.00041

Srikanth G Tamilselvam, Naveen Panwar, Shreya Khare, Rahul Aralikatte, Anush Sankaran, and Senthil Mani. 2019. A Visual Programming Paradigm for Abstract Deep Learning Model Development. In *Proceedings of the 10th Indian Conference on Human-Computer Interaction* (2019-11). ACM, 1–11. DOI:http://dx.doi.org/10.1145/3364183.3364202

Paul Vincent, Kimihiko Iijima, Mark Driver, Jason Wong, and Yefim Natis. 2019. Magic Quadrant for Enterprise Low-Code Application Platforms. (2019).

Marcus Woo. 2020. The Rise of No/Low Code Software Development—No Experience Needed? 6, 9 (2020), 960–961. DOI:http://dx.doi.org/10.1016/j.eng.2020.07.007

Received May 2023; revised September 2023; accepted January 2024