Low-Code Development and End-User Development: (How) Are They Different?

Bernhard Schenkenfelder[1], Ulrich Brandstätter[1], Harald Kirchtag[2] and Manuel Wimmer[3]

[1]*Software Competence Center Hagenberg GmbH, Softwarepark 32a, 4232 Hagenberg, Austria*
[2]*KEBA Group AG, Reindlstraße 51, 4040 Linz, Austria*
[3]*Johannes Kepler University, Altenberger Straße 69, 4040 Linz, Austria*

Abstract

In our daily work, we are often faced with the challenge of designing tools for non-programmers (citizen developers or domain experts) to build their own software, which requires us to investigate appropriate methodologies. At first glance, Low-Code Development (LCD) and End-User Development (EUD) seem like natural candidates. This raises the question of the similarities and differences between LCD and EUD, which this paper explores based on relevant literature. The results show that while both approaches share high-level goals, there are notable differences.

Keywords

Low-Code Development (LCD), End-User Development (EUD), Self-Service Terminal

1. Introduction

In an industry-academia collaboration, we were tasked with developing the front-end and back-end of a self-service terminal (see Figure 1, left) for the public sector. Benefits for the front-end users include (i) the ability to use the terminal outside of office hours, (ii) multimodal interaction such as touch and natural language in the form of speech-to-text and text-to-speech, (iii) interactively guided processes, (iv) safe and secure data through local AI models, and (v) transactions such as printing, receiving and issuing official documents, making payments, and establishing their identity. Overall, the front-end aims to provide a straightforward user experience for its users–the citizens of Austria.

Regardless of whether these processes are handled by a self-service terminal or a counter clerk, they are complex and require frequent updates due to their scope in terms of time, location, and content. For example, heating and electricity subsidies may be introduced and then withdrawn as the economy improves. Similarly, a child care subsidy application may vary from state to state. Finally, a person may be eligible for a driver's license at age 17 instead of 18.

Therefore, not only the front-end, but also the back-end provides self-service capabilities, allowing the public sector domain experts to customize the processes of the terminal. Using a Visual Programming Environment (VPE), which can be learned in a few hours in contrast to months required for a textual programming language, the processes and sequences are represented visually in a flow-based notation, see Figure 1 (center). A dedicated library provides ready-made building blocks that bundle frequently used functionality (Figure 1, right).

In summary, public sector domain experts without software development training (back-end end users) use visual programming (a low-code programming model) to tailor the processes of a self-service terminal for Austrian citizens (front-end end users). The natural question that follows is whether the above case describes Low-Code Development (LCD), End-User Development (EUD), or both, or neither.

Proceedings of the First International Workshop on Participatory Design & End-User Development - Building Bridges (PDEUD2024), October 2024, Uppsala, Sweden
Bernhard.Schenkenfelder@scch.at (B. Schenkenfelder); Ulrich.Brandstaetter@scch.at (U. Brandstätter); ki@keba.com (H. Kirchtag); manuel.wimmer@jku.at (M. Wimmer)
https://www.scch.at/team/bernhard.schenkenfelder (B. Schenkenfelder); https://www.scch.at/team/ulrich.brandstaetter (U. Brandstätter); https://www.keba.com/ (H. Kirchtag); https://se.jku.at/manuel-wimmer/ (M. Wimmer)
0000-0001-5129-6268 (B. Schenkenfelder); 0009-0009-8472-3524 (U. Brandstätter)

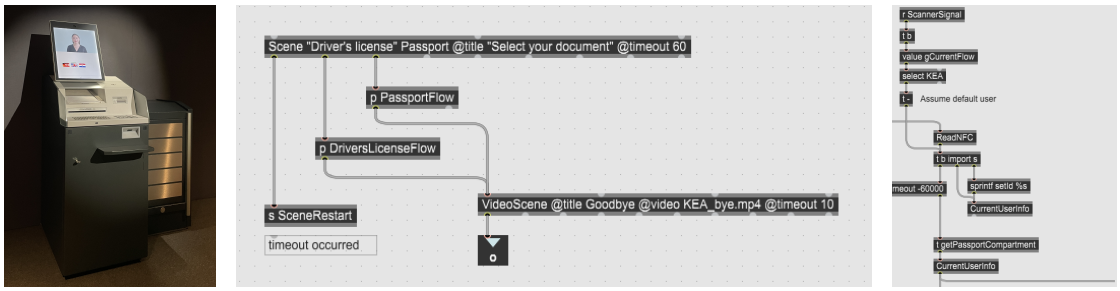

Figure 1: Interactive self-service terminal for citizen services (left). Visual Programming (VP) back-end example: Guide a user to collect a passport or driver's license (center). Details of the *PassportFlow* element of the parent flow (right).

To help clarify this, this paper examines the similarities and differences between the two, based on relevant literature from both fields, without attempting to provide a complete overview. The selected literature largely represents two scientific communities: The authors of [1, 2, 3, 4] are active in the MODELS conference[1] or the co-located LowCode workshop[2], [5] is a very recent and frequently cited low-code paper, and [6, 7, 8, 9, 10] can be associated with the International Symposium on End-User Development (IS-EUD)[3].

In a reference work on EUD, Lieberman et al. [11] use the following definition, which is reused or extended in all of the EUD papers reviewed:

> "End-user development can be defined as a set of methods, techniques, and tools that allow users of software systems, who are acting as non-professional software developers, at some point to create, modify or extend a software artifact."

On the other hand, although all LCD papers use the term low-code, only Pinho et al. [1] propose an explicit and standalone definition:

> "Low-code development is a set of approaches, technologies, and tools that enable rapid application development through techniques that reduce the amount of code written. These approaches can make it possible for end-user developers to program software and use techniques and tools often including but not limited to model-driven engineering, domain-specific languages, and drag-and-drop mechanisms."

These two definitions also suggest a certain similarity or overlap between the two approaches, semantically and in that both can be applied to the above case description. Therefore, we conduct a more in-depth investigation.

2. Low-Code Development (LCD)

Low-Code Development Platforms (LCDPs) are cloud-based platforms for "developers of different domain knowledge and technical expertise" to build and deploy applications using model-driven principles. To compare relevant LCDPs, Sahay et al. [2] first develop a taxonomy of features, including graphical user interface, interoperability support, collaborative development support, and deployment support. They observe that the main components can be grouped into the application modeler, the server side, and external services.

Bock and Frank [3] contribute to this research by uncovering the distinguishing technical features of LCDPs through an analysis of seven LCDPs. They identify common features, namely definitions of data structures, access to external data sources, a GUI designer, the ability to provide basic functional

[1] https://modelsconference.org/
[2] https://lowcode-workshop.github.io/
[3] https://iseud.net/

specifications, a library of standard operations, definitions of roles and user rights, deployment support, and that they are based on model-driven development, as well as occasional and rare features. They point out that LCDPs are sold with the claims that they contribute to the efficiency of software development, and that they are useful for both professional developers and citizen developers.

In a comparison of low-code and model-driven approaches, Di Ruscio et al. [4] reveal the differences that lie behind the (high-level) similarities. LCDPs are often cloud-based platforms, they can be used from a browser, and cover the entire application lifecycle from design to deployment. According to their analysis, citizen developers are the primary users of LCDPs, and most applications fall in the business domain and some in the IoT/event-driven domain.

Pinho et al. [1] conduct a systematic literature review of 38 relevant articles and synthesize common characteristics of LCD: non-programmers as users (13 articles), visual tools with drag-and-drop (12), raised abstractions (9), writing little code (8), presence of underlying models (8), rapid application development (5), lifecycle management (5), and cloud-based infrastructure (5). They also identify the following types of applications being built: business processes (6), databases (6), user interfaces (6), web (5) and mobile (5) applications, and industrial applications (1). Low requirements for technical skills is the most prominent benefit (18) of LCD, followed by short development time (7) and low cost (4).

In a review of the low-code literature, Hirzel [5] notes that low-code users range from professional developers to citizen developers or amateur programmers with little or no software development training. While low-code empowers citizen developers to create computer programs, it makes professional developers more productive. He explains the key building blocks, strengths, weaknesses, and mitigations of three popular low-code techniques: Visual Programming Languages (VPL), Programming by Natural Language (PBNL), and Programming by Demonstration (PBD).

3. End-User Development (EUD)

Paternò [6] points out the main motivation for EUD: Professional developers often lack domain knowledge, while end users find it difficult to communicate their requirements. In addition, these requirements change too frequently for regular development cycles. The success of Web 2.0 also shows that people want to be involved, but they often do not have the means to change the behavior, functionality, and accessibility of the content they create. Therefore, a major goal of EUD is to reduce the conflict between application complexity and learning effort (low threshold vs. high ceiling). Other key concepts are composing and customizing basic elements developed by programmers, collaboration, participatory design, formal and informal representations, and natural language descriptions. To compare EUD approaches, Paternò suggests the dimensions of the generality of the approach (domain-specific vs. general-purpose), the coverage of the main interactive application aspects (interactive vs. functional parts), and whether and how abstractions are used to hide the implementation details.

In a report covering more than 20 years of their EUD research, Myers et al. [7] note that studying the natural ways in which end users describe their tasks can greatly support system design. They also observe that end users and professional developers very often have similar problems–on a different scale. For example, debugging in EUD tools should be designed to help end users find the cause of unwanted behavior.

A systematic mapping study of 165 papers carried out by Barricelli et al. [8] presents approaches and techniques to support end users in tailoring, extending, and creating digital artifacts. Component-based (visual programming) and rule-based (trigger-action rules) techniques are the most common, followed by programming by demonstration/example, spreadsheets, wizards, and templates. As for the application domains, business and data management, web applications and mashups, and smart objects and environments are the most prominent. 65 papers are aimed at domain experts, while 100 papers address the needs of generic users, 6 of which allow advanced development for domain experts. Interestingly, they conclude that more user-oriented methods are needed to further advance the topic.

Fischer [9] explores how AI can help or hinder all EUD stakeholders in addressing their problems. It is important to note that some AI developments may undermine the goals of EUD, as trusting

algorithms can be seen as giving up control and concentrating power. Therefore, Explainable AI (XAI) is an important step in overcoming the black-box nature of AI. Interestingly, and related to low-code, protecting users from low-level code is seen as an area that can benefit from AI. Overall, some examples show that AI and EUD components can be successfully integrated.

Looking at EUD again in the age of IoT and AI, Paternò [10] emphasizes that automation in smart spaces has become part of everyday life, with some systems even making decisions without explicit user request. To avoid unintended actions, it is crucial that users have tools to control these AI-powered systems. Acceptance of an automation is directly related to its transparency to end users, which in turn is determined by its perception, understanding, predictability, and modifiability. Therefore, Paternò suggests that both AI and EUD approaches should coexist and be combined in such tools.

4. Results

First, EUD dates back to scientific work in the 1980s [10], while the more recent term LCD was coined by consulting firms in 2014 [4]. MDE, as a precursor of LCD, has its own scientific community focused on the technical realization of modeling languages, tools, and supporting infrastructures.

The definitions indicate a difference that can be found in most of the papers: EUD focuses on who develops software rather than how they do it as in LCD. Thus, EUD is primarily aimed at citizen developers or domain experts without software development training, while LCD also promises to make professional developers more productive [5].

Interestingly, their (self-perceived) relevance to today's pressing issues varies widely. Fischer [9] advocates research in EUD to improve the quality of life for all people, addressing issues such as democratization and collective creativity. LCD authors, on the other hand, often have a narrower focus on technology and its users [3, 4, 5].

In terms of techniques, the direct manipulation of visual abstractions in an advanced Graphical User Interface (GUI) is the most important in both areas and is found throughout the literature reviewed. Otherwise, there is a strong overlap in additional techniques such as spreadsheets, natural language, and programming by demonstration [5, 8]. While a model-based or model-driven implementation is very often at the core of LCD, it remains unclear what the most dominant technical foundations of EUD are, with model-based [8], meta-design [9], and rule-based [10] as possible candidates.

The scope of tools and infrastructure is also different. Typically for LCD, a cloud-based platform or service covers the lifecycle of a digital artifact from design to deployment, through the abstraction and automation of deployment and provisioning, possibly with the drawback of vendor lock-in [1, 2, 4]. EUD has also moved beyond desktop applications, but it still does not support instant delivery to the same extent [6, 8, 10].

Finally, there are some more differences in the application domains. LCD targets business processes, user interfaces, databases, web and mobile applications, and industrial applications as application types. EUD, on the other hand, also includes smart objects and environments, games and entertainment, education and teaching, and healthcare and wellness. Also, while web and mobile applications are equally important for LCD, there are far more web than mobile applications reported for EUD [1, 8].

5. Conclusion and Future Work

Although LCD and EUD share the same goal of empowering non-programmers to create their own software using tools with a low entrance barrier, there are differences as shown above. However, for our practical work in industry-academia collaborations, both approaches, either alone or in combination, can help achieve this goal, and we can draw from both areas. Returning to the self-service terminal described at the beginning, we could add additional layers of abstraction to address different skill backgrounds, from public sector novices to domain experts, and provide them with a more tailored technological foundation for participatory design of terminal processes.

In future work, it has to be studied if the initial findings generalize and hold true in broader/other contexts. To answer the question posed in the title of this paper, future work will include the application of methods such as a systematic literature review, scientific community analyses, and tool comparisons. Finally, since LCD is used by industry practitioners, exploring their goals and how they view EUD in relation to LCD through individual case studies or more general surveys could further advance this research.

Acknowledgments

The research reported in this paper has been funded by the Federal Ministry for Climate Action, Environment, Energy, Mobility, Innovation and Technology (BMK), the Federal Ministry for Labour and Economy (BMAW), and the State of Upper Austria in the frame of the SCCH competence center INTEGRATE (FFG grant no. 892418) in the COMET - Competence Centers for Excellent Technologies Programme managed by Austrian Research Promotion Agency FFG.

References

[1] D. Pinho, A. Aguiar, V. Amaral, What about the usability in low-code platforms? a systematic literature review, Journal of Computer Languages 74 (2023) 101185.
[2] A. Sahay, A. Indamutsa, D. Di Ruscio, A. Pierantonio, Supporting the understanding and comparison of low-code development platforms, in: 2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), 2020, p. 171–178.
[3] A. C. Bock, U. Frank, In search of the essence of low-code: An exploratory study of seven development platforms, in: 2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), IEEE, Fukuoka, Japan, 2021, p. 57–66.
[4] D. Di Ruscio, D. Kolovos, J. de Lara, A. Pierantonio, M. Tisi, M. Wimmer, Low-code development and model-driven engineering: Two sides of the same coin?, Software and Systems Modeling 21 (2022) 437–446.
[5] M. Hirzel, Low-code programming models, Communications of the ACM 66 (2023) 76–85.
[6] F. Paternò, End user development: Survey of an emerging field for empowering people, ISRN Software Engineering 2013 (2013) 1–11.
[7] B. A. Myers, A. J. Ko, C. Scaffidi, S. Oney, Y. Yoon, K. Chang, M. B. Kery, T. J.-J. Li, Making End User Development More Natural, Springer International Publishing, Cham, 2017, p. 1–22.
[8] B. R. Barricelli, F. Cassano, D. Fogli, A. Piccinno, End-user development, end-user programming and end-user software engineering: A systematic mapping study, Journal of Systems and Software 149 (2019) 101–137.
[9] G. Fischer, End-User Development: Empowering Stakeholders with Artificial Intelligence, Meta-Design, and Cultures of Participation, volume 12724 of *Lecture Notes in Computer Science*, Springer International Publishing, Cham, 2021, p. 3–16.
[10] F. Paternò, End-User Development, Springer International Publishing, Cham, 2023, p. 1–27.
[11] H. Lieberman, F. Paternò, M. Klann, V. Wulf, End-User Development: An Emerging Paradigm, volume 9, 2006, pp. 1–8.

Bernhard Schenkenfelder is a researcher and senior software engineer at the Software Competence Center Hagenberg. He holds an M.Sc. in Business Informatics and is pursuing his Ph.D. from Johannes Kepler University (JKU) Linz, Austria. His primary research interests are low-code development, human-computer interaction, and web-based technologies.

Ulrich Brandstätter is a senior researcher at the Software Competence Center Hagenberg for the Human Centered System Design research focus. His research interests include visual programming

paradigms, experimental interfaces, and game studies.

Harald Kirchtag is the innovation manager at KEBA Group, where he has dedicated over 20 years to advancing the company's technological and business goals. His career at KEBA includes significant leadership roles such as vice president of banking automation and business unit manager for banking automation.

Manuel Wimmer is full professor leading the Institute of Business Informatics–Software Engineering at the Johannes Kepler University Linz. His research interests comprise foundations of model engineering techniques as well as their application in domains such as tool interoperability, legacy modeling tool modernization, model versioning and evolution, and industrial engineering.