Bus Inf Syst Eng
https://doi.org/10.1007/s12599-021-00726-8

DISCUSSION

Low-Code Platform

Alexander C. Bock · Ulrich Frank

Received: 5 February 2021 / Accepted: 10 September 2021

Keywords Low-code · Software development environment · Citizen developer · Organizational agility · Conceptual modeling · Software development productivity

1 Introduction

Under the heading of 'low-code', a new class of software development environments has emerged in recent years which is not only said to afford the prospect of a substantial increase in software development productivity, but also to yield new ways of promoting business IT alignment and user empowerment. These platforms now go by the names of *low-code platform* (LCP), *low-code application platform* (LCAP), and *low-code development platform* (LCDP).

Presumably coined by a market research company in 2014 (Forrester) (Richardson and Rymer 2014), several indicators suggest that a major trend has by now evolved around the label 'low-code'. Large software vendors, including IBM, Microsoft, and Oracle, have begun to incorporate low-code solutions into their product portfolios. Market research companies have forecast a considerable market potential for LCPs (e.g., Rymer and Koplowitz 2019; Vincent et al. 2020). These assessments, together with the promises of vendors, have attracted the interest of corporate investors (Shah 2020). For example, Siemens recently bought a leading LCP vendor, reportedly at a significant price.[1] Moreover, the discussion and presentation of LCPs thrives on, and itself perpetuates, the trends surrounding other current buzzwords such as 'citizen developer', 'robotic process automation' (RPA), 'user experience', and 'microservices'.

Unfortunately, the rise in attention has not been paralleled by similar advances in the conceptualization of LCPs. For initial reference and for later comparison we cite, at this stage, a definition proposed by a market research firm:

> "An LCAP is an application platform that supports rapid application development, deployment, execution and management using declarative, high-level programming abstractions such as model-driven and metadata-based programming languages, and one-step deployments. LCAPs provide and support user interfaces (Uls), business processes and data services." (Vincent et al. 2020, p. 1)

An analysis of this tentative characterization of LCPs suggests two conclusions. First, LCPs are intended to help achieve objectives which have been at the core of business information systems research for a long time. Among these are *increasing productivity* and *reducing costs* of developing and maintaining enterprise software systems, improving organizations' ability to *adapt* software systems to rapidly changing requirements, and *empowering users*. Second, however, it is not at all clear what distinguishes LCPs from existing software development facilities, such as classical integrated development environments (IDEs) and tools for model-driven development (MDD). Taken together, the low-code trend presents itself as a

Accepted after one revision by Christof Weinhardt.

A. C. Bock (✉) · U. Frank
Research Group for Business Informatics and Enterprise Modeling, University of Duisburg-Essen, Essen, Germany
e-mail: alexander.bock@uni-due.de

[1] https://www.forbes.com/sites/adrianbridgwater/2018/08/06/siemens-buys-low-code-mendix-the-digital-factory-race-climbs-higher/. Accessed 12 Sep 2021.

Published online: 15 November 2021

contribution to goals at the heart of business information systems research, but is in dire need of clarification.

With this in mind, the goal of this paper is to give a balanced account of the current trend of low-code development, and to place the topic in the broader context of business information systems research. Specifically, we address three questions:

1. What are the characteristic features of low-code platforms?
2. How do low-code platforms compare with the current status of research, and what, if any, technological innovations are realized by these platforms?
3. What opportunities for future research arise from the present attention to low-code development?

The paper is structured as follows. We begin with a brief survey of the reception of the low-code trend in practice and academia (Sect. 2). Subsequently, we present an answer to the first question, drawing on a detailed study of ours of selected LCPs available on the current market (Sect. 3). The study was decidedly exploratory and, given the limited number of systems considered, cannot claim to be representative, but the obtained results nonetheless throw light on the characteristic features of a variety of LCPs. Thereafter, we turn to the second question, formulating several general findings and contrasting them with existing approaches from research on software development productivity (Sect. 4). These findings, in turn, provide the basis for examining the third question, leading us to suggest a variety of attractive research opportunities for the field of business information systems (Sect. 5).

2 Aspects of the Trend: Promises and Reception

The trend surrounding low-code development is carried by an appealing story. It is widely known that the lack of professional software developers is a major obstacle for many companies in successfully dealing with the digital transformation. Moreover, there is the perennial problem that software development projects often suffer from poor efficiency, or fail altogether. Portrayals of LCPs by vendors and market research firm tie in with these problems and promise relief:

> "When you can visually create new business applications with minimal hand-coding – when your developers can do more of greater value, faster – that's low-code."[2]
>
> "That's where the power of citizen development comes in – with no-code/low-code platforms, anyone can build applications without software expertise, significantly faster, and at a fraction of the cost."[3]
>
> "Enterprise low-code application platforms deliver high-productivity and multifunction capabilities across central, departmental and citizen IT functions." (Vincent et al. 2020, p. 1).

The enthusiasm and hopes engendered by claims such as these are further nurtured by reports of successful applications of low-code solutions in industry (e.g., Shah 2020), and by optimistic assessments of the future economic potential of the low-code sector. For example, Gartner has predicted that by "2023, over 50% of medium to large enterprises will have adopted an LCAP as one of their strategic application platforms" (Vincent et al. 2020, p. 1).

It was only after some years that academic investigators took notice of this trend (e.g., Henriques et al. 2018; Zolotas et al. 2018). But now there is growing awareness of low-code in the research community. For example, the workshop on low-code development at the *Models* conference in 2020 attracted the highest number of submissions of all workshops.[4]

Regrettably, much work on low-code development to date has assumed a relatively uncritical attitude, and all too little effort has been spent on developing a clear and distinct concept of these systems. For example, one author has defined LCPs thus: "The low-code platform is a set of tools for programmers and non-programmers. It enables quick generation and delivery of business applications with minimum effort to write in a coding language and requires the least possible effort for the installation and configuration of environments, and training and implementation" (Waszkowski 2019, p. 376). The favorable, even euphoric, tone and the conceptual ambiguity of this statement are characteristic of many descriptions found in the field (cf., e.g., Chang and Ko 2017; Ihirwe et al. 2020; Sanchis et al. 2020). That said, a modicum of work towards a more nuanced view of low-code development is scattered in the literature. For example, Sahay et al. (2020) have conducted a study of eight LCPs with the aim of clarifying and comparing the functionalities of these systems. Moreover, there are a few critical voices that question the supposed novelty of low-code environments. The most notable example is Cabot (2020); we will return to a verdict of his later on.

This brief outline of the reception of low-code in practice and in academia indicates a propensity toward marketing jargon, terminological inexactitude, and inflated promises. Meanwhile, profit and non-profit organizations face the question of whether, and under what conditions,

[2] https://www.ibm.com/uk-en/automation/low-code. Accessed 12 Sep 2021.

[3] https://www.pmi.org/citizen-developer. Accessed 12 Sep 2021.

[4] https://lowcode-workshop.github.io/. Accessed Sep 12 2021.

investments into low-code development projects are justified. The need to support these decisions is a further reason for business information systems research to analyze the subject and to contribute to a clarification of the term.

3 In Search of a Conceptualization: Results from an Exploratory Study

In want of informative and consistent descriptions of LCPs, the only way to obtain a concept or even an idea of LCPs is an analysis of the actual platforms offered under this label on the current market. If these platforms have common features, it may become possible to derive inductively from them a possible conceptualization of low-code development. With this aim in mind, we conducted an explorative study of ten LCPs, which has been published separately (for a synopsis with a technical focus, see Bock and Frank 2021; for the full study, see Frank et al. 2021). The purpose of this section is to summarize the central results of this study. The reader interested in further details is referred to the references cited.

There are now between about twenty and several dozen vendors selling products under the label 'low-code.' It was, of course, beyond our scope to undertake an exhaustive review of all these solutions, so that a limited number of LCPs had to be selected for study. Accordingly, no claim of representativeness can be made. Because the study is intended to be explorative in nature, however, we argue that this restriction is justifiable. A number of criteria were considered in the selection of LCPs. The overriding intent was to cover a spectrum as broad as possible. We therefore began with a preliminary market study, grouping existing LCPs into four rough categories according to their *general character and purpose* (cf. below). Afterwards, we chose the LCPs so that at least one candidate from each category was considered. Moreover, we made sure to cover vendors of different *size* and *market influence*, as well as LCPs intended for different *target audiences*. In case of doubt, large vendors were given the priority, based on the assumption that these figure prominently in shaping the concept of LCPs. More details on the selection process are found in the original study. Limitations of the analysis and related work are indicated at the end of this section; and for the complete details, again, see the references cited above.

3.1 Historical Background and Product Positioning

A striking fact about almost all considered low-code products is they have existed, in one form or another, *before* the label 'low-code' was invented. With the exception of one platform, the solutions now marketed as LCPs have been the single or primary products of the offering companies for years, if not decades. An analysis of archived versions of the vendors' homepages revealed that most products have been further developed and repositioned over the years. For example, some systems have previously been offered as solutions for 'rapid application development', 'platform as a service' (PaaS), and, in one case, as a 'model-driven application platform'. Another platform has a history as a tool for 'business process management' (BPM) in general, and as a tool for 'mobile' BPM and 'human-centric' BPM in particular. The bottom line is that present-day low-code solutions are rarely new products; more often than not, they have a long history on the market and have simply been rebranded.

Another significant finding is that platforms available in the low-code sector *vary drastically* in several dimensions. Although certain technical features are common to many LCPs, as will be discussed in the next subsection, the studied environments differ substantially in functional scope, primary purpose, range of technologies involved, breadth of applicability, means of design and specification, and other respects. For example, the four rough categories of LCPs we distinguished in the pre-study included (1) simple *data management platforms*, (2) classical *workflow management systems* (WfMS), (3) *extended, graphical user interface (GUI)- and data-centric IDEs*, and (4) *complex multi-use platforms* for business application configuration, integration, and development (for a fuller description, see Bock and Frank 2021). The upshot is that platforms offered under the label 'low-code' are exceedingly heterogeneous and do not constitute a well-defined class of technological environments.

3.2 Features of Low-Code Platforms

We now turn to the identified features of the studied LCPs, which we will present according to their frequency of occurrence, distinguishing between *common*, *occasional*, and *rare* features. Where relevant, we will assign these features to the traditional perspectives of systems development – the *static*, the *functional*, and the *dynamic* perspective. A summary is given in Fig. 1.

Common features. Most common features of LCPs fall into the static perspective. Every considered product features a component for the definition of *data structures*. This component is almost always offered in the form of a *conceptual modeling tool*, implementing either a classical data modeling language (i.e., a variant of the Entity-Relationship Model) or a simplified proprietary language. Sometimes, data structures can only be defined in UI-based dialogs or lists. A related common feature is the capacity to access *external data sources* using a variety of application programming interfaces (APIs). For example, the platforms all permit using standard APIs like JDBC, as well as

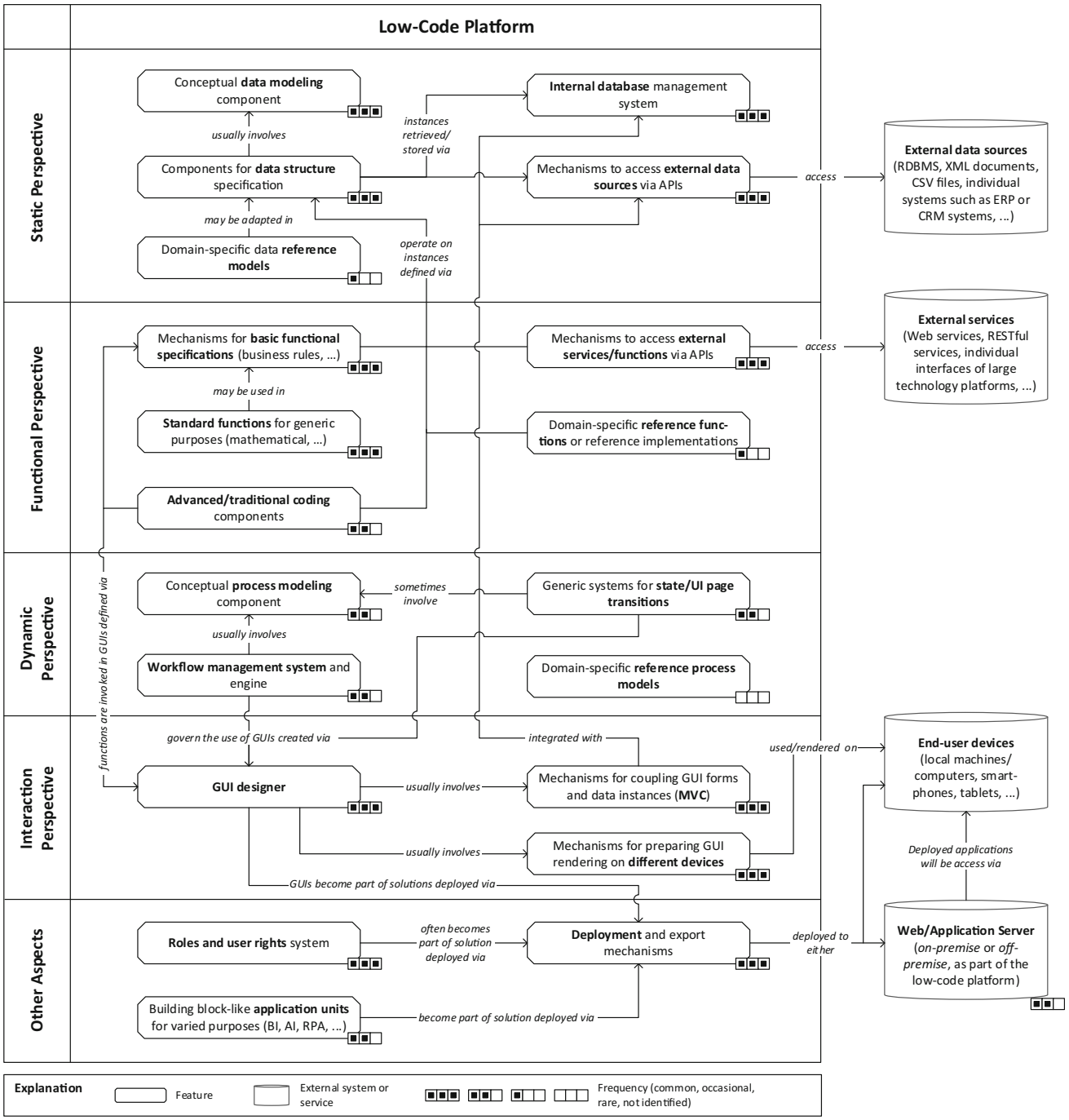

Fig. 1 Features of low-code platforms

various connectors to other types of files and systems. As a rule, the platforms are designed so that data may be stored either in an *internal database system*, or in existing *external systems*.

Another feature provided by every studied low-code platform is a *GUI designer*. Without exception, the studied platforms incorporate a component to develop graphical user interfaces and to integrate them with other implementation artifacts. All reviewed GUI designers bring palettes of pre-defined widgets, although their scope varies widely. Defining the *coupling of GUIs and data structures* is fairly convenient in most environments, as it is not necessary to implement the model-view-controller (MVC) pattern manually. Furthermore, most systems provide distinct support in adapting the GUIs to *different target environments* (e.g., desktop browsers, tablets, and smartphones).

Furthermore, all systems afford *basic functional specifications*. The most common approach here are simple expression languages for decision rules and dialog-based ways of specifying program flow conditions. Similarly, each solution provides a library of generic *standard operations*, such as mathematical functions. Also, all solutions enable, in varying ways, to invoke and integrate *external functions* via *APIs*. For example, almost every system allows to use standard approaches such as web services and RESTful (representational state transfer) services; and many systems make it possible to use a long list of APIs of individual providers, such as Google APIs and other social media APIs.

Moving on to a different area, almost all considered solutions offer advanced support in the *deployment of the applications*, although this takes quite different forms. For example, some systems require to install the environment of the low-code platform on a web server, so that individual applications can be deployed there. Other systems, in contrast, allow to deploy the developed solutions as self-contained applications on various devices and machines. Finally, another common feature is that almost all considered solutions came with a component to define *roles and user rights*. The roles and user system is usually contained in the governing architecture of the platform, which will be deployed together with the custom application.

Occasional features. Many less common features fall into the dynamic perspective. One occasional feature is the availability of a *workflow modeling component* and a *workflow engine*. Several of the larger and more complex LCPs incorporate a WfMS at their architectural core; other systems now branded as LCPs, as previously indicated, really *are* classical WfMS. The platforms either use a conceptual modeling language like Business Process Model and Notation (BPMN), or a proprietary representational structure. In other systems, the perspective is more generic and technical. These platforms mainly provide components to define the *interaction with, and transition between, user-defined UI forms*. Some platforms rely on proprietary process modeling component for this purpose, others on basic menus and event-catching scripts.

Another occasional feature is the availability of advanced or *traditional coding components*. Some systems, involve one or several explicit components where procedural specifications can be made using traditional programming languages and related technologies. Most frequently, the systems use Java and JavaScript. Indeed, at some more or less hidden level of the architecture, almost *all* low-code platforms grant recourse to traditional programming code. Finally, the studied platforms of major vendors offer a variety of *configurable, building block-like application units*. For example, these units provide, in limited scope, pre-implemented functionalities for the areas of business intelligence (BI), artificial intelligence (AI), and RPA.

Rare features. Notably, domain-specific reference implementation artifacts were identified only on rare occasions. Only one large-scale platform provides a considerable library of *domain-specific reference data models*, addressing common business and communication concepts, such as ‘customer’, ‘address’, and ‘email’. Some other systems offer small sets of simple reference structures, and most reviewed LCPs do not bring with them any data reference models at all. When it comes to *domain-specific reference functions* and *reusable artifacts of a dynamic nature*, still fewer artifacts are found. One large-scale platform makes it possible to reuse a number of ready-made functions from various business applications of the same vendor. Most other studied systems offer catalogs of reusable functions or examples of predefined processes, but these are mostly generic in nature, or very limited in scope and depth.

3.3 Limitations and Related Work

A number of limitations apply to our study. The most important one has already been emphasized at the outset: because of the limited sample size, no claim to representativeness can be defended. Another general limitation is that besides the various features highlighted above, every solution has, of course, a range of individual features, which cannot be enumerated here. Several more specific limitations concern the fact that there are a number of criteria whose analysis was unfeasible for us. This includes, for example, the behavior of the LCPs in the deployment and scalability of large projects, as well as cost-efficiency analyses. To our knowledge, the only other study of available LCPs is that of Sahay et al. (2020). The aims and results of that study are similar to ours, but Sahay et al. (2020) concentrate on comparing particular LCPs, whereas our interest is to reveal and critically discuss the main features of LCPs in the context of business information systems research. For a more detailed discussion of the results and limitations of our analysis, see Bock and Frank (2021) and Frank et al. (2021).

4 Discussion and Assessment

Our exploratory analysis of existing low-code platforms has revealed a number of features found in these systems, and it has occasioned several general observations. We will now discuss the main findings of our analysis, offering a critical assessment of the trend.

Low-code platforms integrate various classical development components in one environment. The most important way in which the examined LCPs differ from classical software development infrastructures is that they incorporate all or most tools and components required for a limited class of software development projects in a single environment. When using a classical software development infrastructure, one has to deal with a sizeable array of separate tools, such as IDEs, modeling tools, database management systems (DBMS), object-relational (O-R) mapping frameworks, GUI editors, deployment and compilation assistants, and so on. In LCPs, these tools are integrated into one system, so that there is less need to switch between different systems and, more importantly, less need to keep consistent and integrate the implementation artifacts produced by the distinct technologies.

Reuse is addressed at a generic architectural level, not at a domain-specific level. The investigated low-code platforms rarely offer reusable artifacts at a domain-specific level. Rather, what is reused by these platforms are fairly generic, and often implicit, architectural frameworks for certain classes of application systems. This involves reference implementations for such areas as GUI design, O-R mappings, MVC operations, access to external data sources and other services, and so on. Within these frameworks, then, individual solutions can be configured and developed.

Productivity gains mainly ensue from reducing the efforts of routine tasks. LCPs produce productivity gains primarily by reducing the efforts of routine tasks in software development projects of low to moderate complexity. This applies in several ways. One is that through the provision of a pre-defined, integrated environment, there is less effort in synchronizing the artifacts produced by previously separate development components. Productivity is also promoted by the aforementioned reference implementations for such generic tasks as GUI design, O-R mapping, MVC implementation, and deployment in different environments.

No new technology. Summing up the previous points, we arrive at the finding that most components of LCPs are, in and of themselves, neither radically new nor innovative in any way. To the contrary, what is actually provided in these environments are well-known tools and components for software development which have been used, in varying forms, for decades. Also, many low-code products have, in fact, existed for many years, and are now merely rebranded. In this respect, we therefore agree with Cabot who concluded: "I do not believe there is any fundamental technical contribution in low-code trend" (Cabot 2020, p. 536).

Conceptual modeling is not at the core of marketing, but at the core of the platforms. Another key finding concerns not the technology itself, but the way in which it is sold. As our analysis has indicated, *conceptual modeling components* are among the most important components of low-code platforms and one of the principal ways in which they are able to decrease the need for traditional coding. This is the exact approach of the field of MDD, where the vision has been that conceptual models are used to "reduce the gap between problem and software implementation domains through the use of technologies that support systematic transformation of problem-level abstractions to software implementations" (France and Rumpe 2007, p. 37). The modeling languages provided by the LCPs, however, are rather simplistic, lagging behind the state of the art in research on conceptual modeling.

Incomplete account of related research. The design of available LCPs has, evidently, derived inspiration from several lines of research on IS design and implementation, but, surprisingly, ignored others. A field from which most LCP vendors have drawn quite extensively is, as underlined above, that of *model-driven development*. There are also more or less pronounced similarities to other research-based approaches, although the exact technical concepts and procedures are not usually adopted. This is true, for example, of design constructs from research on 'visual programming' (e.g., Costagliola et al. 2004; Ingalls et al. 1988), technical concepts to support 'weaving' reusable services into customized ones (e.g., Besova et al. 2012; Bergel and Fabry 2009), and the more recent vision of 'on-the-fly computing' (Karl et al. 2020). Remarkably, other well-known lines of investigation have been ignored altogether. Most importantly, as already stressed, this concerns work on *reference models* (e.g., Becker and Delfmann 2007; Fettke and Loos 2007) as well as work on *domain-specific modeling languages* (e.g., Kelly and Tolvanen 2008; Frank 2013). Both reference models and DSMLs can promote the productivity of systems design significantly, as they incorporate domain knowledge.

Risk of lock-in effects. In most cases, the representations generated in one LCP cannot be used in another. This presents a serious threat to the protection of investment and may lead to dead ends when a vendor stops supporting a platform (as is the case with *AppMaker*, which Google has recently abandoned).

5 Opportunities for Future Research

A number of opportunities for research are tied to the emergence of low-code development platforms. One reason is that methods and theories for the construction and adaptation of organizational information systems are at the very core of business information systems research. Another reason is that all areas of our social reality

continue to be permeated by software. Thus, much in the spirit of low-code development, approaches are needed to empower people – not just the traditional "user" – to understand and modify the software shaping their work environment and, more and more, their entire lives. The following list of research opportunities is intended to illustrate the scope of inspiring research topics related to the low-code trend.

Support for the evaluation and economic use of low-code platforms. As has become clear, LCPs are no silver bullets, but, under the appropriate circumstances, the use of these platforms can help organizations improve productivity and agility in software design and development. It is, however, far from trivial to judge the economics of LCPs. Research is needed to identify problem classes which can be effectively addressed by the typical functions of LCPs – and those which cannot. It is also essential to investigate whether and how LCPs can serve as a supplement to, rather than a replacement of, traditional software development infrastructures. Furthermore, the support of organizational decision makers calls for research on the training required for employees with no or little programming skills to use LCPs effectively, as well as on the costs and risks of this sort of institutionalized lay development. Following on from this, another line of research could be directed toward the design and evaluation of methods for the economic analysis of LCPs, as well as (modeling) methods for guiding lay developers in the use of LCPs and classical software development facilities.

New approaches to study the possibilities and limitations of domain-specific reference models: As our analysis has shown, domain-specific reference models and equivalent implementation artifacts are rarely provided in low-code platforms. This raises the question of why this is so, as reference models could easily be integrated into these environments. Reference models may not only reduce development costs and enable a higher quality of information systems, they may also promote (cross-organizational) integration. But it is also true that domain-specific reference models, despite some enthusiasm a few decades ago, did not become the game changers they were supposed to be. The emergence of LCPs is a welcome occasion for business information systems research to (re-)address itself to the obstacles preventing the use of reference models and the design of strategies to overcome them.

Cognitive fit of representations: Although LCPs address both professional software developers and 'citizen developers', it remains unclear what kind of user models they employ. Such models are needed to make representations of software fit cognitive capabilities and personal working styles. The investigation of this topic opens a wide range of research questions whose investigation may involve fields such as cognitive psychology, and modeling and programming language design.

Effects on organizational behavior: Approaches to foster end-user computing effectively blur the traditional boundaries between developers and users. This is likely to affect organizational decision processes, patterns of collaboration, and roles surrounding IS design and use. While these matters have been the subject of early studies on end-user computing (Amoroso, 1988), the digitization of work environments as well as the average levels of computer literacy have changed considerably since then. This leads to research questions concerning the future conception of IT management in general, and the organization of software projects in particular.

6 Conclusion

Our analysis of low-code platforms did not produce evidence that the individual components of low-code solutions are radical innovations. In many ways, they lag behind the frontiers of research on software design, implementation, and maintenance. What distinguishes these platforms is that they integrate, in one environment, multiple well-known and traditional system design components so as to reduce the efforts of routine tasks in implementing business applications within the confines of certain, more or less restrictive frameworks. As such, LCPs can promote software development productivity *if* all the requirements of a given project can be satisfied within the predefined, immutable framework of a certain LCP *and* all other pertinent technical and economic conditions are fulfilled, too. Developing methods for the careful assessment of these conditions is an important subject for future research.

Moreover, the momentum generated by the low-code trend gives rise to various other inspiring research opportunities lying not only at the core of our discipline, but also at the cross-sections with other disciplines. A most notable opportunity lies in the fact that the attention directed at low-code platforms may contribute to the revival of *conceptual modeling*. As we have shown, conceptual modeling is one of, if not *the* single most important ingredient of present-day low-code platforms – even though vendors rarely declare themselves as offering modeling environments. Business information systems research is well positioned to seize this opportunity, since it is the only discipline that takes the design and analysis of domain-specific conceptual models as one of its fundamental tasks.

The term 'low-code', however, is problematic. It is currently used in an inconsistent manner, being deployed to sell vastly heterogeneous development environments. While we do not think it is advisable to prescribe

practitioners what terminology to use, we believe that it is our responsibility to critically reflect on whether terms are suitable for incorporation into a proper technical terminology. We do not think the term 'low-code' at present satisfies the criteria required of a scientific concept. After all, language is our most important tool, and we should beware of compromising it.

Funding Open Access funding enabled and organized by Projekt DEAL.

References

Amoroso DL (1988) Organizational issues of end-user computing. SIGMIS Database 19(3–4):49–58. https://doi.org/10.1145/65766.65773

Becker J, Delfmann P (eds) (2007) Reference modeling: efficient information systems design through reuse of information models. Physica, Heidelberg

Bergel A, Fabry J (eds) (2009) Software composition. In: Proceedings 8th international conference on software composition, Zurich. Springer, Heidelberg

Besova G, Walther S, Wehrheim H, Becker S (2012) Weaving-based configuration and modular transformation of multi-layer systems. In: France RB et al (eds) 15th international conference on model driven engineering languages and systems, innsbruck. Springer, Heidelberg, pp 776–792

Bock AC, Frank U (2021). In search of the essence of low-code: an exploratory study of seven development platforms. In: Proceedings of the 24th ACM/IEEE international conference on model driven engineering languages and systems: companion proceedings. ACM and IEEE, New York

Cabot J (2020). Positioning of the low-code movement within the field of model-driven engineering. In: Proceedings of the 24th ACM/IEEE international conference on model driven engineering languages and systems: companion proceedings. ACM and IEEE, New York, pp 535–538. https://doi.org/10.1145/3417990.3420210

Chang YH, Ko CB (2017) A study on the design of low-code and no code platform for mobile application development. Int J Adv Smart Convergence 6(4):50–55. https://doi.org/10.7236/IJASC.2017.6.4.7

Costagliola G, Deufemia V, Polese G (2004) A framework for modeling and implementing visual notations with applications to software engineering. ACM Trans Softw Eng Meth 13(4):431–487. https://doi.org/10.1145/1040291.1040293

France RB, Rumpe B (2007) Model-driven development of complex software: a research roadmap. In: Briand LC, Wolf AL (eds) Workshop on the Future of Software Engineering. IEEE, New York, pp 37–54. https://doi.org/10.1109/FOSE.2007.14

Frank U (2013) Domain-specific modeling languages – requirements analysis and design guidelines. In: Reinhartz-Berger I et al (eds) Domain engineering: product lines, conceptual models, and languages. Springer, Heidelberg, pp 133–157

Frank U, Maier P, Bock AC (2021) Low code platforms: promises, concepts and prospects. A comparative study of ten systems. ICB Research Report 70. University of Duisburg-Essen, Essen

Henriques H, Lourenço H, Amaral V, Goulão M (2018) Improving the developer experience with a low-code process modelling language. In: Proceedings of the 21st ACM/IEEE international conference on model driven engineering languages and systems. ACM, pp 200–210. https://doi.org/10.1145/3239372.3239387

Ihirwe F, Di Ruscio D, Mazzini S, Pierini P, Pierantonio A (2020) Low-code engineering for internet of things. In: Proceedings of the 23rd ACM/IEEE international conference on model driven engineering languages and systems: companion proceedings. ACM, pp 522–529. https://doi.org/10.1145/3417990.3420208

Ingalls D, Wallace S, Chow YY, Ludolph F, Doyle K (1988) Fabrik: a visual programming environment. SIGPLAN Not 23(11):176–190. https://doi.org/10.1145/62083.62100

Karl H, Kundisch D, Meyer auf der Heide F, Wehrheim H (2020) A case for a new IT ecosystem: on-the-fly computing. Bus Inf Syst Eng 62(6):467–481. https://doi.org/10.1007/s12599-019-00627-x

Kelly S, Tolvanen JP (2008) Domain-specific modeling. enabling full code generation. Wiley, Hoboken

Fettke P, Loos P (eds) (2007) Reference modeling for business systems analysis. Idea, Hershey

Richardson C, Rymer JR (2014) New development platforms emerge for customer-facing applications. Forrester Report, Forrester

Rymer JR, Koplowitz R (2019) The Forrester wave: low-code development platforms For AD&D professionals, Q1 2019. Forrester Report, Forrester.

Sahay A, Indamutsa A, Di Ruscio D, Pierantonio A (2020) Supporting the understanding and comparison of low-code development platforms. In: Proceedings of the 46th Euromicro conference on software engineering and advanced applications. IEEE, New York, pp 171–178. https://doi.org/10.1109/SEAA51224.2020.00036

Sanchis R, García-Perales Ó, Fraile F, Poler R (2020) Low-code as enabler of digital transformation in manufacturing industry. Appl Sci 10(1):12. https://doi.org/10.3390/app10010012

Shah A (2020). Emptying offices prompt adoption of low-code to build work apps. In: The Wall Street Journal, 15 May 2020. https://www.wsj.com/articles/emptying-offices-prompt-adoption-of-low-code-to-build-work-apps-11589535001. Accessed 12 Sep 2021.

Vincent P, Natis Y, Iijima K, Wong J, Ray S, Jain A, Leow A (2020) Magic quadrant for enterprise low-code application platforms. Gartner Report September 2020, Gartner

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

Zolotas C, Chatzidimitriou KC, Symeonidis AL (2018) RESTsec: a low-code platform for generating secure by design enterprise services. Enterp Inf Syst 12(8–9):1007–1033. https://doi.org/10.1080/17517575.2018.1462403