Viljoen, Altus; Hein, Andreas; Krcmar, Helmut

Low-code development platform ecosystems

Electronic Markets

Provided in Cooperation with:
Springer Nature

Suggested Citation: Viljoen, Altus; Hein, Andreas; Krcmar, Helmut (2026) : Low-code development platform ecosystems, Electronic Markets, ISSN 1422-8890, Springer, Berlin, Heidelberg, Vol. 36, Iss. 1,
https://doi.org/10.1007/s12525-025-00848-x

Electronic Markets (2026) 36:12
https://doi.org/10.1007/s12525-025-00848-x

FUNDAMENTALS

Low-code development platform ecosystems

Altus Viljoen[1] · **Andreas Hein**[2] · **Helmut Krcmar**[1]

Received: 5 March 2025 / Accepted: 13 October 2025

Abstract
Low-code development platforms (LCDPs) such as Mendix, OutSystems, and Microsoft Power Platform are reshaping software development. These platforms enable non-technical users to develop applications by reusing and configuring out-of-the-box components in visual, user-friendly environments. While LCDPs are typically portrayed as new standalone development tools, they also exhibit many characteristics of digital platform ecosystems. For example, similar to traditional app stores, they also offer marketplaces where external developers and business users can contribute what they have developed. Yet, despite these conceptual similarities, the "platform" dimension of LCDPs has received little attention. Accordingly, this Fundamentals paper conceptualizes LCDPs as a distinct type of digital platform ecosystem. We build on research in digital platform ecosystems, boundary resources, packaged software reuse, and platform governance, and explain how LCDPs align with and differ from conventional platforms. Specifically, we show how LCDPs combine infrastructure and reuse, embed design constraints directly into components, and orchestrate ecosystem participation through a logic of "curated enablement" rather than open contribution. We conclude by outlining a research agenda that highlights LCDPs not as marginal tools for end-user development, but as rich, theory-relevant settings for further exploration.

Keywords Low-code development platform · No-code · Citizen development · Software reuse · Digital platform ecosystem

JEL Classification L86 · O31 · O33 · O36 · O39

Introduction

Low-code development platforms (LCDPs), such as Mendix, OutSystems, and Microsoft Power Platform, offer visual, user-friendly environments and pre-built components that facilitate rapid application development (Bock & Frank, 2021). Due to their user-friendly nature, these platforms are regarded as transformative technologies that can "democratize" software development by enabling individuals with limited or no programming expertise to build functional applications (Carroll et al., 2024; Sahay et al., 2020). LCDPs are increasingly recognized as essential tools for organizations, with 80% of companies noting them to be key for accelerating software development efforts (KPMG, 2024).

While LCDPs are often described as standalone development tools, their core infrastructure—integrated development environments (IDEs) and pre-built components—is opened up to a broader ecosystem of users and developers. Providers such as Mendix and OutSystems not only supply IDEs but also create mechanisms for external actors to extend platform functionality. For example, the Mendix Marketplace (2025c) and OutSystems Forge (2025a) let third-party developers publish reusable low-code components, which others can then integrate into new applications. This follows the well-established logic of digital platform ecosystems. Development environments are exposed to external developers—for example, software development kits (SDKs) in Android or iOS—who create complementary modules that expand the platform's value (Hein et al., 2020; Tiwana et al., 2010). Importantly, like traditional digital platform

Responsible Editor: Mark de Reuver

✉ Altus Viljoen
altus.viljoen@tum.de

[1] TUM School of Computation, Information and Technology, Technical University of Munich, Boltzmannstr. 3, 85748 Garching, Germany

[2] Institute for Information Systems and Digital Business, University of St. Gallen, Müller-Friedberg-Strasse 8, 9000 St.Gallen, Switzerland

ecosystems, LCDPs not only enable third-party development but also provide curated distribution channels. For example, marketplaces and repositories ensure that components are shared, discovered, and recombined across the ecosystem (Hein et al., 2019; Henfridsson & Bygstad, 2013).

However, despite clear parallels to established digital platform ecosystems, the "platform" dimension of LCDPs remains conceptually underdeveloped. This is especially true when considering how these platforms align with or depart from traditional digital platform ecosystems. For example, in contrast to transaction or innovation platforms (Cusumano et al., 2019; Evans & Gawer, 2016), LCDPs operate primarily in proprietary, enterprise-focused environments. They target non-technical users and rely on curated component reuse rather than generic modular contributions. As a result, core digital platform ecosystem concepts—such as boundary resources (Ghazawneh & Henfridsson, 2013), ecosystem governance (Wareham et al., 2014), and generativity (Fürstenau et al., 2023)—must be reinterpreted to reflect the context-specific nature of LCDP ecosystems. Yet, most research on LCDPs has focused on technical features (Bucaioni et al., 2022; Prinz et al., 2021), organizational adoption (Carroll et al., 2024; Viljoen et al., 2024a), or developer behavior (Al Alamin et al., 2023; Elshan et al., 2024). This leaves platform-level and ecosystem dynamics largely unexplored.

In this Fundamentals paper, we address this gap by examining how LCDPs function as digital platform ecosystems. Particularly, we review what distinguishes their infrastructure, reuse mechanisms, and orchestration dynamics from traditional digital platform ecosystems. We show that LCDPs cannot be understood through existing digital platform archetypes alone, as they blend visual development tools, component-based reuse, and enterprise-grade orchestration into a tightly curated yet extensible ecosystem. Our conceptualization builds on four literature streams—digital platform ecosystems (Hein et al., 2020; Tiwana et al., 2010), boundary resources (Ghazawneh & Henfridsson, 2013), packaged software reuse (Vial, 2023), and platform governance (Wareham et al., 2014).

We identify three elements that distinguish LCDP ecosystems. First, LCDP infrastructure is not only extensible (Tiwana et al., 2010) but also supports what we call *layered generativity* (Fürstenau et al., 2023). This means that innovation happens across multiple layers: the infrastructure provides basic capabilities, developers contribute reusable components, and those components can themselves be combined into new applications that extend the infrastructure. Second, reusable low-code components are curated, modular assets that drive software reuse across applications. However, unlike generic code modules, *low-code components embed platform-specific design logic*, making them easier to assemble visually but also constraining how they can be adapted and combined. Third, LCDP ecosystems are orchestrated through what we describe as *curated enablement*. Here, platform owners open participation to a wide range of users but maintain strong control over tools, workflows, and standards (Wareham et al., 2014). This differs from traditional digital platform ecosystems, where layers tend to be more loosely connected and communities have more autonomy (Yoo et al., 2010).

Our conceptualization of LCDPs as a distinct class of digital platform ecosystems contributes in two ways. First, it advances digital platform ecosystem research by showing how LCDPs reshape core concepts such as modularity, generativity, and governance. In LCDPs, reuse is not a downstream efficiency practice but a foundational design principle, generativity is distributed across layers rather than concentrated in a single locus, and governance takes the form of curated enablement rather than open contribution. Together, these dynamics challenge the assumption that openness and generativity must always coincide (Cennamo & Santaló, 2019; Wareham et al., 2014). Second, our framework provides a foundation for future empirical research into LCDP dynamics. This includes boundary resource design, control–participation trade-offs, and the evolution of reuse-based value creation in enterprise software environments.

Theoretical foundation

To conceptualize LCDPs as a distinct form of digital platform ecosystem, we build on four streams of literature: (1) digital platform ecosystems, (2) boundary resources, (3) packaged software reuse, and (4) platform governance. We briefly review each stream and its core assumptions. We later show how these assumptions regarding infrastructure extensibility, reuse, and governance must be reinterpreted to capture the unique dynamics of LCDP ecosystems.

Digital Platform Ecosystems. Digital platforms are commonly understood as extensible software infrastructures that mediate interactions between producers and consumers of digital content (De Reuver et al., 2018; Tiwana et al., 2010). Central to this view is the notion of modular architecture: a stable platform core (e.g., operating system, cloud infrastructure) is extended by peripheral modules developed by third parties (Baldwin and Woodard, 2009; Yoo et al., 2010). In this model, platform owners offer third parties the ability to extend the platform by contributing apps, plug-ins, or services that extend the value of the platform or add consumable value to the ecosystem (Hein et al., 2020).

Boundary Resources. To enable such third-party extensions, platform owners rely on boundary resources: the standardized means through which access to the platform is structured and controlled (De Reuver et al., 2018; Ghazawneh & Henfridsson, 2013). Boundary resources make

platforms modular and extensible by exposing selected functionalities while safeguarding the stability of the core infrastructure (Wareham et al., 2014). In traditional digital platform ecosystems in the software development domain (e.g., Android and iOS), boundary resources typically take the form of SDKs and application programming interfaces (APIs).

Packaged Software Reuse. In digital platform ecosystems in the software development domain, packaged software reuse plays a key role and complements boundary resources. Boundary resources such as SDKs and APIs are examples of pre-built, standardized components, packaged for easy reuse and integration. These software packages enable developers to (re)combine artifacts without building every feature from scratch. Developers therefore rely heavily on reuse, and see their role as writing "glue-code" that integrates existing components (Sojer & Henkel, 2010; Vial, 2023). Packaged software reuse can thus be understood as the practical manifestation of boundary resources, but it emphasizes a different logic. Boundary resources define *how* access is granted and controlled, while reuse highlights *why* developers depend on such access for efficiency. In this way, boundary resources structure access, but their value becomes tangible when developers use pre-packaged components to build working applications.

Platform Governance. Platform governance research emphasizes the tension between openness and control in platform ecosystems (Cennamo & Santaló, 2019; Wareham et al., 2014). Platform owners must strike a balance between enabling third-party innovation and maintaining architectural coherence, quality, and strategic direction. This tension is typically addressed through modular boundaries, certification regimes, or tiered developer programs (Staub et al., 2023). Governance decisions thus shape who can participate, how contributions are made, and how value is distributed within the ecosystem (Tiwana et al., 2010). In traditional software development ecosystems, governance mechanisms are closely tied to modular boundaries, i.e., platform owners define standardized interfaces (e.g., APIs, SDKs) that allow extension but constrain deviation (De Reuver et al., 2018). Such mechanisms not only regulate technical integration but also establish rules, roles, and incentives that influence the evolution of the ecosystem over time (Chen et al., 2022). As a result, platform governance becomes a key lever through which platform owners sustain innovation, mitigate risks, and pursue strategic goals.

Low-code development platforms

Before applying insights from the four literature streams to conceptualize LCDPs as digital platform ecosystems, it is necessary to establish a clearer understanding of LCDPs. This section, therefore, offers a descriptive overview of typical LCDP types, outputs, and target users. The overview draws on academic literature and industry sources and is intended to provide a descriptive understanding of LCDPs, rather than an interpretive or theoretical perspective, which follows in the subsequent sections.

LCDPs[1] are software environments that enable users to build applications through visual interfaces and reusable components, minimizing the need for traditional programming (Bock & Frank, 2021; Sahay et al., 2020). While LCDPs typically require minimal traditional coding to provide users the opportunity to customize pre-built components, some platforms aim to remove coding completely. Such platforms are referred to as **no-code development platforms** (Viljoen et al., 2025). LCDPs vary widely in scope and capability, from simple tools for data handling to complex enterprise-grade environments supporting workflow automation and cross-platform app development (Bock & Frank, 2021; Richardson & Rymer, 2014; Vincent et al., 2020). Generally, four types of LCDPs can be distinguished (Bock & Frank, 2021), as shown in Table 1.

Given the breadth of LCDPs, the types of outputs they deliver vary considerably. LCDPs like Mendix and OutSystems enable the creation of comprehensive web and cross-platform mobile applications. Workflow management services like Airtable, Zapier, and Microsoft Power Automate are designed primarily for workflow automation and resemble robotic process automation technologies (Eggers et al., 2023). Additionally, some LCDPs focus specifically on creating websites (e.g., Wix, Shopify, Microsoft Power Pages), while others focus on dashboards that support data analytics and business intelligence (e.g., Airtable, Microsoft Power BI). Moreover, LCDPs offering these capabilities do not do so in a mutually exclusive manner and often provide a mixture of such offerings. For example, LCDPs like ServiceNow's Now Platform and Microsoft Power Platform offer a central platform with comprehensive capabilities, but they are adapted based on use cases or licensing for the enterprise.

Nevertheless, despite their differences, the common denominator is that LCDPs are not used to develop complex, large-scale software systems. Instead, they typically support applications created within business units by users without formal programming expertise. These developers are often referred to as **citizen developers** (Gartner, 2024) or **business unit developers** (Scharpf et al., 2024). For

[1] LCDPs are also synonymously referred to as *low-code platforms (LCPs)* or *low-code application platforms (LCAPs)* (Bock & Frank, 2021).

Table 1 Types of low-code development platforms (adapted from Bock & Frank, 2021)

Platform type	Description	Examples
Basic data management platforms	Allows users to structure and manage data through user-friendly interfaces, abstracting away technical database concepts. These platforms focus on collaborative data handling and configuration	Airtable, Power BI
Workflow management services	Focuses on visual workflow design and automation, enabling users to model processes, interfaces, data schemas, user roles, and system integrations using drag-and-drop tools	Zapier, Microsoft Power Automate
Extended GUI- and data-centric IDEs	Integrates interface design, data modeling, and business logic into a unified low-code environment. Aimed at more intermediate to advanced users, these platforms streamline web and app development, though complex features may still require conventional programming	Mendix, OutSystems, Microsoft Power Apps
Multi-use platforms for business applications	Typically offered by major vendors and designed with enterprise integration in mind. These platforms enable the configuration of applications using modular components and pre-built assets and aim for seamless alignment with existing enterprise systems and broader IT ecosystems	ServiceNow, Microsoft Power Platform

example, employees at a bottling company used Microsoft Power Platform to automate backend processes for managing vending machines (Microsoft, 2020). Similarly, workers at a healthcare provider developed a flu vaccine administration app using ServiceNow's Now Platform (ServiceNow, 2024).

Elements of low-code development platform ecosystems

With a descriptive overview of LCDPs established, we now examine the underlying elements that explain LCDPs and the structure and dynamics of their surrounding ecosystems. We identify three interdependent elements: (1) technical infrastructure, (2) reusable low-code components, and (3) platform owner ecosystem governance. This examination is based on an interpretative synthesis of existing literature and platform documentation. It provides the foundation for the subsequent theoretical conceptualization of LCDP ecosystems and their divergence from adjacent theoretical concepts.

Technical infrastructure of LCDP ecosystems

LCDP infrastructure comprises the technical backbone that enables LCDP ecosystems. We distinguish between platform-owner[2] infrastructure and third-party infrastructure.

LCDP owner infrastructure: layered architecture, IDEs, and component marketplaces

While LCDPs differ in scope and complexity (Table 1), they are based on a similar **layered architecture** and share many functionalities (Sahay et al., 2020).[3] Different platforms incorporate these layers to varying degrees, depending on their intended purpose. These functionalities are offered through **user-friendly IDEs** that provide a visual, drag-and-drop interface for application design (e.g., Mendix Studio Pro, OutSystems Service Studio). They are complemented by **component marketplaces** (e.g., Mendix Marketplace, OutSystems Forge), where users can upload, discover, import, and integrate pre-built components. Some components are embedded within the IDE and can be used "out-of-the-box," while others are accessed via the marketplace and then customized in the IDE. Table 2 summarizes the four layers and provides examples of components available to platform users within each layer.

The component marketplaces—as with the LCDPs themselves—differ significantly, making a one-to-one comparison of the marketplaces and the hosted reusable components challenging. Nevertheless, we identify consistent attributes, as shown in Table 3.

[2] In the context of LCDP ecosystems, the "platform owner" refers to the firm that offers the IDE, accompanying infrastructure, and orchestrates the ecosystem. Examples are Mendix, OutSystems, and ServiceNow.

[3] For a detailed overview of LCDPs and their common functionalities, we refer the reader to Bock and Frank (2021) and Sahay et al. (2020).

Table 2 Layered architecture of LCDPs (adapted from Sahay et al., 2020)

Layer	Description	Examples of components
Application	The top layer where users interact with the graphical user interface (UI), build applications and their UIs, and where users specify application behavior through logic and workflows	Buttons, text fields, dashboards, drag-and-drop elements for defining conditionals and loops
Service integration	The layer where external services are integrated into the application, typically connected through APIs and authentication mechanisms	API connectors (e.g., with social media), authentication modules
Data integration	The layer where data from various sources and in different formats (e.g., spreadsheets, cloud storage) is integrated and manipulated	Database connectors, extract-transform-load tools, data format converters
Deployment	The layer where applications are deployed on cloud infrastructures or on-premise environments. It manages the containerization and orchestration of applications, and connects with the service integration layer for continuous integration and deployment	Containerization tools, e.g., Docker containers or Kubernetes pods

Third-party infrastructure: Cloud-based services

In addition to IDEs and marketplaces, LCDP ecosystems rely heavily on third-party infrastructure. These external services and platforms extend development, integration, and deployment capabilities beyond what LCDP owners provide. Cloud service providers like Amazon Web Services and Microsoft Azure are especially critical for deploying and scaling low-code applications. For example, Mendix supports deployment to private cloud environments through integrated pipelines that leverage GitOps and container tooling (Mendix, 2025b). Third-party services also play a key role in code and version-control functionalities. OutSystems, for instance, integrates directly with GitHub through continuous integration and continuous deployment connectors, allowing developers to push build manifests and deployment logs to repositories (OutSystems, 2025b).

Reusable low-code components

Reusable low-code components comprise the second element shaping LCDPs and the structure and dynamics of their surrounding ecosystems. At a basic level, LCDPs rely on pre-built components—ranging from form fields to database connectors—that can be visually configured and combined to create applications. These components are often described as the "building blocks" of low-code platforms. OutSystems (2024), for example, notes that "components are one of the most important aspects of low-code platforms, as they enable code reusability and repositories of component libraries … [enabling] developers [to] build high-quality app components and share them with others."

Such components are essential because they allow developers to avoid creating every feature from scratch; instead, developers can assemble applications through configurable and reusable modules. Yet, components differ widely across platforms: some are small and granular, while others are comprehensive or domain-specific. To capture these variations systematically, we identify five dimensions of reusability, which we group into **backend** and **front-end** dimensions. We distinguish between backend and front-end dimensions to make this model more generalizable across LCDPs. The backend dimensions concern *how and where reusable functionality is technically enabled,* while the

Table 3 Common attributes of low-code components in LCDP marketplaces

Attribute	Description
Component type	Refers to the "granularity" of components. This includes categorizations such as widgets (e.g., single-purpose components such as buttons), templates (e.g., components encompassing a collection of widgets), or comprehensive pre-built applications
Category	Refers to the "purpose" of the component. This may include "purpose categorizations" directly corresponding to the LCDP architectural layers (e.g., whether the component is intended for the UI, data integration, etc.) or refer to another broad functional purpose (e.g., artificial intelligence)
Industry	Pertains to the specific sectors or fields that the component is tailored for, like manufacturing, healthcare, finance, retail, etc
Compatibility	Indicates which releases or versions of the LCDP the component is compatible with
Support	Denotes whether the component is supported by the LCDP owner, the community, or other third parties
Price	Outlines the pricing structure for the component, i.e., whether the component is free or commercially licensed

front-end dimensions capture *how components are packaged and experienced by users*. This distinction is important because the same technical functionality can be instantiated in very different ways for end users. For example, a data connector may be offered as an industry-agnostic functionality in the backend, but be packaged within an industry-specific module in the front-end.

Backend reusability

Reuse levels correspond to the four architectural layers (Table 2) at which reusable functionality is made available, with LCDPs typically supporting reuse across the UI, logic, data, and deployment layers. At the *UI level* (application layer), reuse encompasses visual components—such as buttons, layout templates, and navigation widgets—that determine an application's presentation. For example, OutSystems (2025a) offers a dedicated category of reusable UI components in its Forge marketplace. At the *logic level*, functionality reuse involves application behavior, including reusable flows, rules, and control structures. Logic reuse can span both the application layer (when provided directly in the GUI by the platform provider) and the service integration layer (when accessed through APIs). For instance, Mendix offers configurable "flow types" that allow developers to model sequential logic, decision trees, or event-driven actions without writing code, enabling them to incorporate pre-built behavior instead of specifying every step manually. At the *data level* (data integration layer), reusable functionality includes pre-modeled schemas and connectors that abstract external data sources. For instance, in OutSystems, this appears as reusable data entities that can be dropped into an application and bound to existing enterprise data with minimal configuration. Finally, at the *deployment level* (deployment layer), reuse encompasses artifacts for packaging, version control, and environment configuration, such as OutSystems' reusable deployment pipelines that allow developers to move applications between test and production environments without setting up new infrastructure.

Reuse enablement describes how LCDP providers make reusable functionalities possible across the different reuse layers discussed above. We distinguish between two reuse enablement types: *Proprietary-based enablement* includes functionalities developed by the LCDP owner—such as Mendix's built-in data models or OutSystems' default UI libraries—which are embedded in the IDE and ready for use. *Third-party-based* enablement, by contrast, involves integrations with external providers that are provided as part of low-code components within the platform. For example, Microsoft Power Platform offers connectors to services such as Salesforce or Dropbox directly inside its IDE, enabling developers to reuse functionalities not developed by Microsoft.

Front-end reusability

While reuse levels and reuse enablement describe the technical provision of components on the backend, users experience them as *instantiated artifacts on the front-end*. These artifacts vary in granularity, scope, and domain-specificity, shaping how reuse is realized in practice for developers.

Component granularity refers to the breadth of functionality that a component offers. Some components are highly granular and serve a single purpose, such as basic widgets or buttons. Others, including templates, modules, or frameworks, combine multiple widgets or functionalities. Component granularity thus illustrates how backend reuse dimensions—reuse levels and reuse enablement—are *packaged for the user* in an intuitively reusable form. Granularity also varies significantly across platforms, making it difficult to compare LCDPs and their marketplaces directly. For example, Mendix Marketplace distinguishes between a wide spectrum of components—from basic widgets such as buttons to multi-widget templates and complete pre-built applications. Microsoft AppSource, by contrast, does not provide lower-level components like widgets in its marketplace, even though such elementary widgets are built into the applications and are available in the LCDP's IDE.

Reuse scope concerns whether a reusable component can be used within a single application or across multiple applications. *Within-application* reuse may include layout elements or shared data models that promote consistency across screens or workflows but remain scoped within a single application. *Cross-application* reuse, by contrast, refers to frameworks or themes that support standardization across multiple development projects. Mendix's Atlas UI framework (2025d) is an example of cross-application reuse: it provides a centralized design language and styling toolkit that can be reused across multiple applications to ensure visual and functional coherence. In practice, this means a developer can automatically reuse the same visual standards across an entire app portfolio.

Finally, **domain specificity** describes how general or industry-focused a component is. Many LCDP components are intentionally generic and *domain-agnostic*—such as login modules or approval workflows—making them usable across diverse contexts. Others are specialized and address *domain-specific* needs, such as in finance, healthcare, manufacturing, or retail. ServiceNow's App Store and Microsoft AppSource, for instance, provide modules tailored for regulatory compliance, patient management, or inventory tracking. For example, in a healthcare setting, such modules can

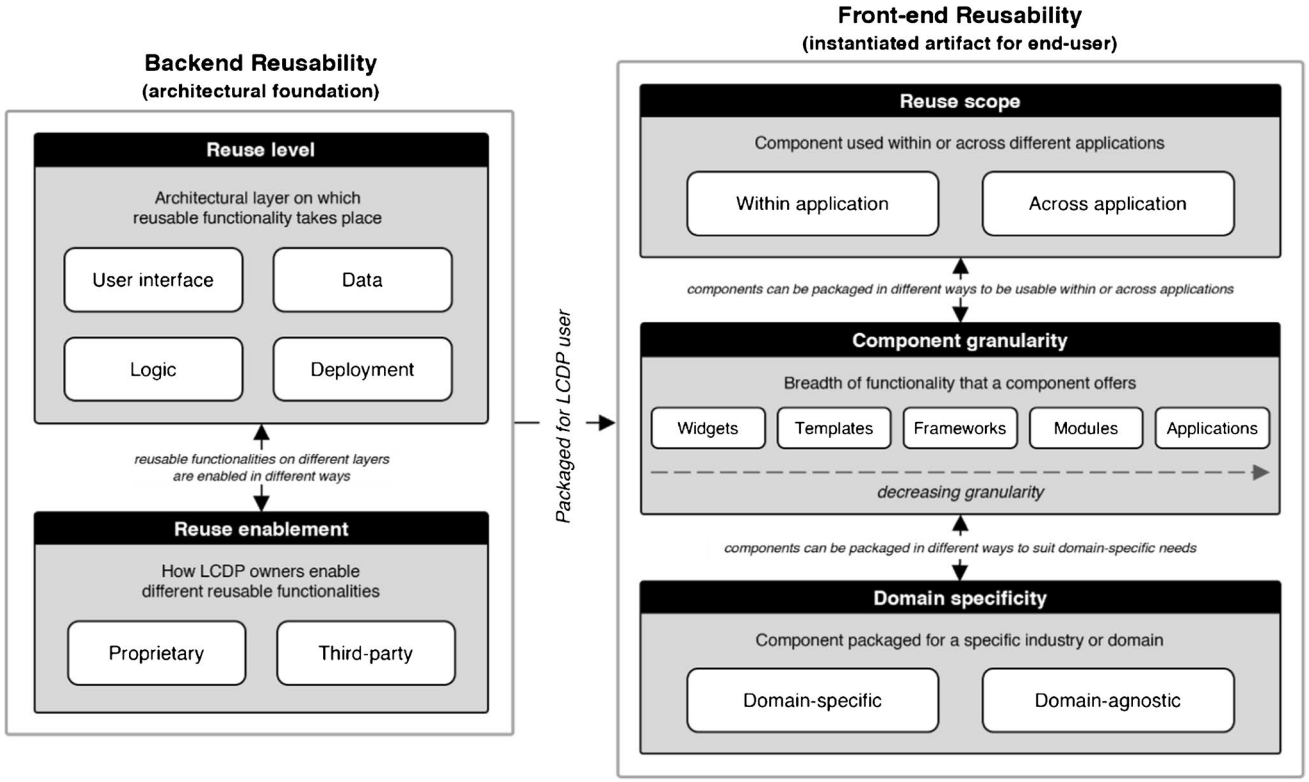

Fig. 1 Dimensions of reusability in low-code components

be utilized by hospitals to deploy a pre-built patient scheduling component instead of building one from scratch. Figure 1 shows a visual synthesis of the five reuse dimensions that characterize the reusability of low-code components.

LCDP owner ecosystem governance

The final element shaping the structure and dynamics of LCDPs and their ecosystems is the governance role of the platform owner. To conceptualize LCDP ecosystems and theorize how they differ from other platform models, it is essential to understand how platform owners orchestrate and govern interactions within the ecosystem.

The dynamics of how LCDP providers, as platform owners, govern ecosystems are shaped by three interrelated factors. First, the **dual role** that the platform owner plays must be considered, as platform owners are responsible for providing both the core infrastructure and publishing reusable components. Second, ecosystem orchestration depends on the **types of actors** the platform owner engages (i.e., platform users and third-party vendors). Third, these **dynamics evolve over time**, as platform owners must not only enable development through infrastructure and components, but also sustain the ecosystem through continuous support. Based on these factors, we identify four governance trends that capture these dynamics. Figure 2 provides a synthesis of these trends.

LCDP owners as infrastructure providers

Optimal integration: Integrating capabilities from third-party providers LCDP owners are responsible for building and maintaining integrations with third-party providers, while developers rely on these connections to create useful applications. Owners *enable* this by linking external services on the backend and exposing them as reusable front-end components. For instance, developers may wish to connect applications with cloud storage services offered by Amazon Web Services or Microsoft Azure, and the platform owner must ensure that these integrations are readily available and easy to implement. LCDP owners also continuously *support* this process by curating, updating, and expanding the range of providers and integrations over time. For example, with the rise of generative artificial intelligence (AI), many LCDPs now integrate access to popular large language models, giving developers streamlined access to capabilities from providers like OpenAI or Google.

User compatibility: Offering interoperability for customers While optimal integration emphasizes backend

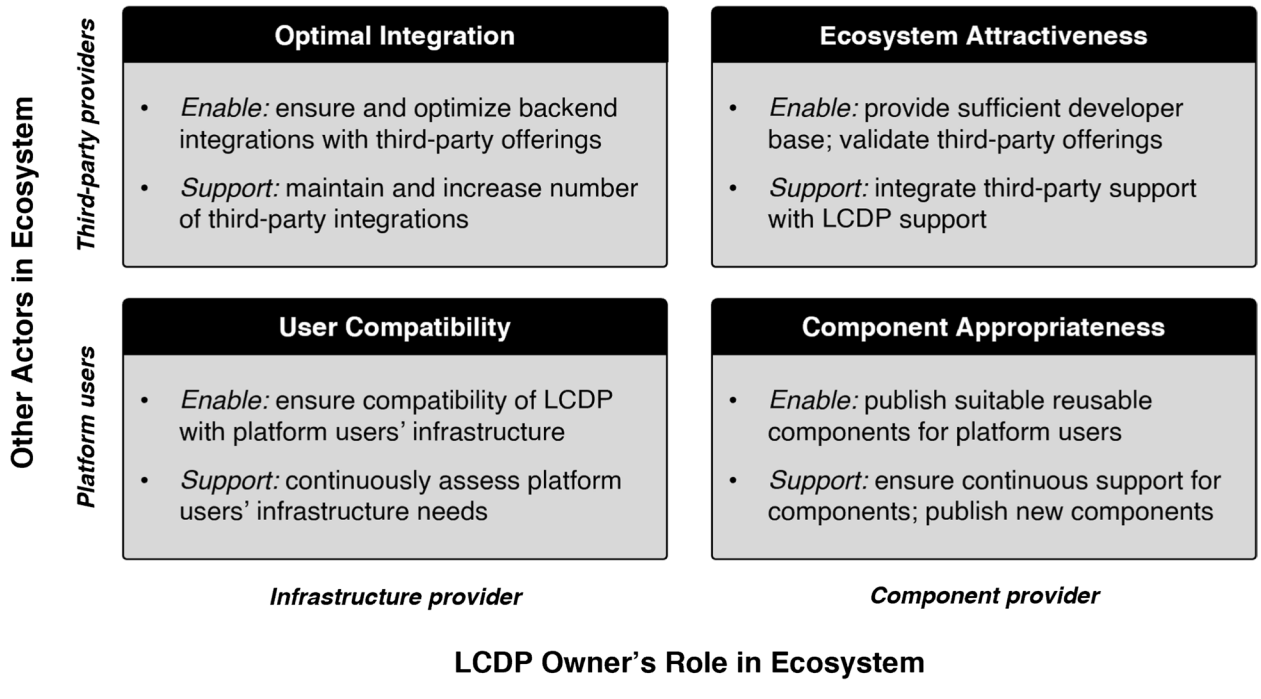

Fig. 2 Trends explaining LCDP ecosystem governance

interoperability with third-party providers, user compatibility is customer-focused and concerns how well the LCDP fits into platform users' existing technology environments. As the focus of many major providers is on enterprises—such as the LCDPs from Mendix, OutSystems, and ServiceNow—LCDP owners ensure that their platforms are compatible with such enterprise information technology (IT) landscapes. Platform users depend on this compatibility to integrate low-code applications easily within their infrastructure. LCDP owners *enable* this by supporting interoperability with both external services that a customer may use (e.g., from cloud providers) and internal systems, such as on-premises infrastructure and open-source technologies. LCDP owners further *support* compatibility by continuously monitoring technology updates and deprecations, adapting their platforms over time to remain aligned with both proprietary and open-source systems that their users make use of.

LCDP owners as reusable component providers

In addition to providing the core infrastructure, LCDP owners also design and facilitate reusable components that developers rely on to build applications efficiently. Two key governance trends emerge in this role.

Ecosystem attractiveness: Motivating third-party providers to develop and publish components LCDP owners are responsible for cultivating ecosystems that attract and retain third-party providers to contribute high-quality components. In turn, LCDP users benefit from the broader choice of such components. LCDP owners *enable* this by implementing mechanisms that make the marketplace appealing to vendors. For example, OutSystems offers certification and validation tiers through its Forge Marketplace, giving accredited components higher visibility and credibility. Such measures serve as an official seal of approval and incentivize vendors to participate. LCDP owners then *support* this attraction over time by updating and expanding validation schemes, ensuring they remain relevant as the ecosystem evolves. For instance, Mendix boosts the credibility of selected providers by including their components directly in official Mendix Docs (Mendix, 2024), which helps maintain vendor visibility as the ecosystem grows.

Component appropriateness: Designing useful components for platform users LCDP owners are responsible for publishing reusable components that align with user needs, while users depend on these components to accelerate their development processes. LCDP owners *enable* this by packaging components so that they are pre-built to save time, yet malleable enough to avoid excessive customization. While certainly not the only measure, this "sweet spot" of malleability can be seen in the download numbers: For example, on Mendix Marketplace (2025a), generic templates for web applications are downloaded far more often than industry-specific solutions. Owners then *support* component appropriateness over time in two ways. First, they continuously publish new components to expand the range of available

building blocks and respond to emerging customer needs. Second, they ensure that existing components remain functional and up to date, often involving the developer community in maintenance. For instance, some Mendix components are community-maintained: developers patch them for security, extend their features, or adapt them to API changes. Together, these mechanisms ensure that components stay relevant and useful as technologies and customer requirements evolve.

Interdependence between LCDP governance trends

These four governance trends capture LCDP ecosystem dynamics through the interplay of who drives the change, and who is affected. Platform owners typically drive governance trends—by expanding integrations, ensuring compatibility, attracting third-party providers, or publishing components—while developers and platform users respond to them. Yet, while each trend has a "primary" stakeholder focus, none operates in isolation. For example, ensuring user compatibility with enterprise IT landscapes also reinforces ecosystem attractiveness, since third-party vendors are more likely to invest in platforms with strong enterprise adoption. Similarly, expanding optimal integrations with external services enhances component appropriateness, as platform users can consume those services directly through pre-packaged building blocks. Thus, these trends show that governance in LCDP ecosystems cannot be reduced to isolated functions but must be understood as a coordinated effort across the ecosystem.

Conceptualizing LCDPs as digital platform ecosystems

Conceptualization and definition

Having described LCDP ecosystems and their structural elements, we now develop a conceptual model of LCDPs that positions them as a distinct class of digital platform ecosystems. Figure 3 presents the conceptual model, synthesizing the relationships between these elements and the roles of various actors.

As shown in the model, reusable low-code components serve as the boundary resource that binds the ecosystem. Similar to SDKs or APIs, these low-code components transfer design capability to users and enable external actors to co-create value (Eaton et al., 2015; Ghazawneh & Henfridsson, 2013; Von Hippel & Katz, 2002). Low-code components thus act as modular artifacts that mediate interaction

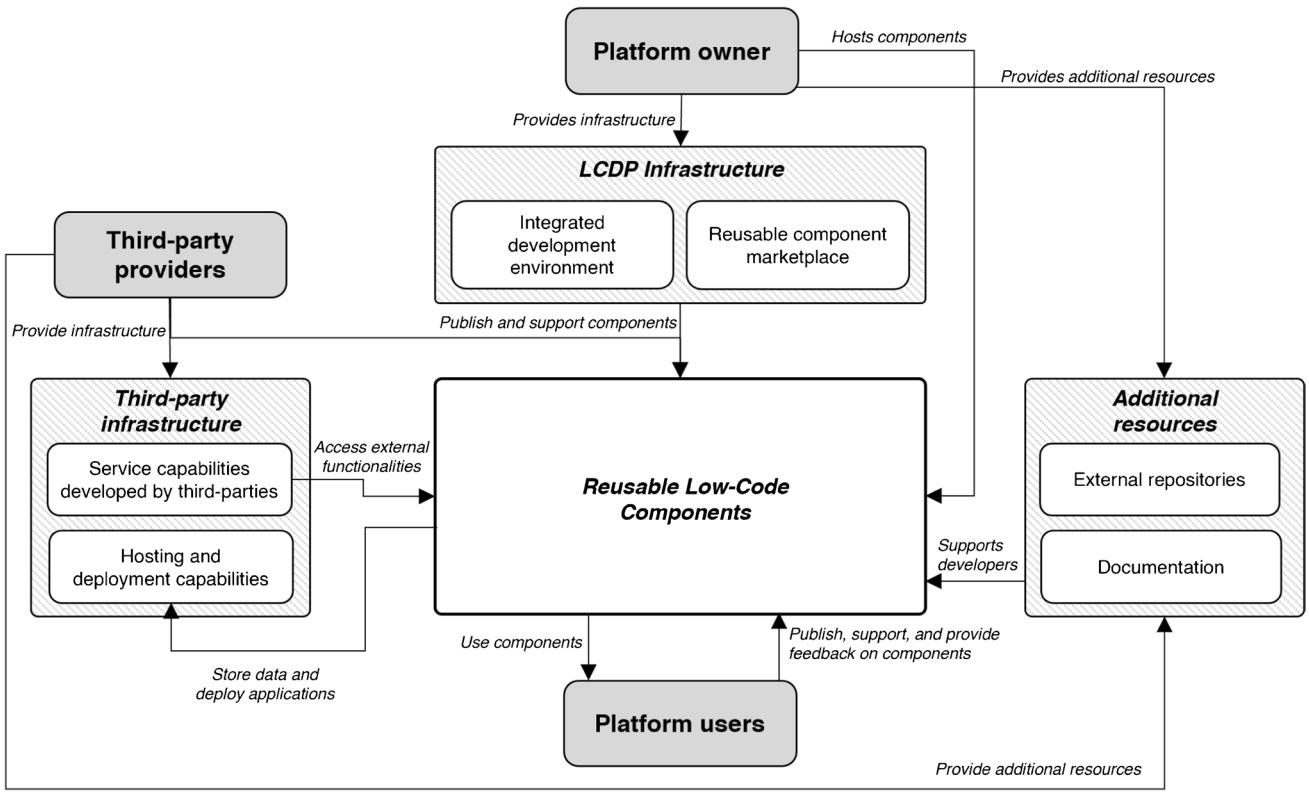

Fig. 3 Conceptual model of low-code development platform ecosystems

between platform owners, platform users, and third-party providers.

Thus, based on our conceptualization, we formally define LCDP ecosystems as follows: *LCDP ecosystems are curated digital platforms that enable scalable application development by turning reusable low-code components into boundary resources. These components are created, shared, and recombined through visual programming environments and governed marketplaces, allowing developers to not only build applications but also extend the ecosystem's generativity through reusable contributions.*

However, while rooted in general digital platform ecosystem principles such as modularity, extensibility, and third-party participation, LCDP ecosystems diverge in how they structure and scale innovation. In the following sections, we further develop our conceptualization and definition of LCDP ecosystems by examining how they align with—or diverge from—adjacent concepts. We structure our conceptualization around the three structural elements of LCDP ecosystems, and integrate the four literature streams introduced in the theoretical background. For each of the three elements, we highlight both parallels and points of distinction.

Theoretically positioning and distinguishing LCDP ecosystems

Technical infrastructure of LCDP ecosystems

LCDPs share many similarities with modern digital platform architectures. Like cloud-based platform-as-a-service offerings, they abstract away low-level hardware concerns and provide a managed, multi-tenant environment for application deployment and scaling (Di Ruscio et al., 2022). They also exhibit a layered modular architecture (Yoo et al., 2010), with a stable core of services and tools and a flexible periphery of extensible modules such as pre-built components (Hein et al., 2020; Tiwana et al., 2010). In this way, LCDP infrastructures mirror the digital backbones of enterprise platforms. They combine storage, integration, and runtime layers into shared technical foundations that support multiple actors (Lusch & Nambisan, 2015). In addition, they adopt technologies from contemporary software engineering—such as containerization and automated orchestration—that enable rapid, one-click deployment. However, LCDPs' infrastructure differs from conventional platform models in several ways. We highlight three fundamental differences below.

Layered generativity: Turning extensions into reusable inputs In traditional digital platform ecosystems in the software development context, extensibility typically means that developers produce extensions as outputs (Tiwana et al., 2010). In other words, applications or services are delivered as "final" outcomes to end-users. In LCDP ecosystems, by contrast, extensions take the form of reusable components that serve as inputs for further development. We call this **layered generativity**: innovation is distributed across successive layers, as infrastructure provides base capabilities, reusable components add modular options, and applications built from these components can themselves be reused or extended. Thus, LCDPs function as generative environments (Fürstenau et al., 2023), where components are created, shared in curated marketplaces, and recombined *across* layers of reuse (Weber et al., 2024). This layered dynamic challenges conventional assumptions about platform innovation and calls for a broader understanding of what it means to "extend" a platform.

Embedded composition: Integrating tools, marketplaces, and workflows In traditional platforms, infrastructure is usually accessed externally through APIs or SDKs—for example, a mobile developer calls Google Maps APIs from outside the platform core to embed location services in an app. In LCDPs, by contrast, development is embedded directly within a visual IDE. We call this **embedded composition**: the integration of tools, marketplaces, and workflows into a single environment that both enables and constrains what can be built. Thus, the IDE, together with the platform-curated marketplace, forms a controlled exchange where reusable components are assembled, configured, and deployed within the same infrastructure. In this way, LCDP infrastructure functions not merely as a toolkit but as a design substrate that orchestrates modularity, interaction, and deployment within defined boundaries.

Proprietary control: Enforcing platform-specific environments Many traditional digital platform ecosystems, such as Android, consist of open infrastructures that foster modular innovation through autonomous, community-driven contributions (Ghazawneh & Henfridsson, 2013; Henfridsson & Bygstad, 2013). In contrast—while some LCDPs promote open standards and allow custom code (e.g., Java or JavaScript in Mendix)—much of their core environments remain proprietary. Mendix applications, for example, must run on their proprietary runtime and are primarily built in their visual IDEs, while many marketplace components are curated or governed by the platform owner. Thus, although LCDPs incorporate selective openness, they strongly rely on proprietary control to enforce consistency, maintain standardization, and deliver enterprise-grade reliability (Viljoen et al., 2025).

Reusable low-code components

Reusable low-code components in LCDPs share many similarities with established notions of boundary resources and

packaged software reuse. In traditional digital platform ecosystems, APIs and SDKs expose stable interfaces to core functionalities, enabling third-party developers to build extensions and co-create value (Eaton et al., 2015; Ghazawneh & Henfridsson, 2013). Likewise, low-code components—such as connectors, plug-in widgets, or templates—mediate interaction between LCDP providers and users (developers). This logic mirrors open-source and enterprise software engineering, where developers reuse modular libraries and packages and primarily write "glue code" to assemble them (Sojer & Henkel, 2010; Vial, 2023). Yet, low-code components depart from conventional notions of packaged software as boundary resources in two important ways, as discussed below.

Endogenous boundary resource creation: Moving from provider-designed to ecosystem-created In digital platform ecosystem and software reuse literature, boundary resources are typically assumed to be designed and provisioned by the platform owner to connect internal capabilities with external contributions (Star, 2010; Tiwana et al., 2010). While ecosystem actors may influence their evolution (Eaton et al., 2015), they remain primarily provider-driven. LCDPs challenge this assumption. Increasingly, boundary resources are created *endogenously*—that is, from within the ecosystem itself—by third parties or even business users who publish reusable components to shared marketplaces. For example, a business user might develop and share a banking module on Mendix Marketplace that other organizations can download, adapt, and reuse. These components are thus not only consumed, but also reused as generative inputs for further development. In this way, boundary resources dynamically evolve as elements co-created within the ecosystem.

Embedded design constraints: Shifting from efficiency to participation and coordination In conventional platforms, reuse is often framed as a matter of development efficiency—developers save effort by reusing pre-built resources such as libraries or SDK functions. In LCDPs, by contrast, reuse additionally becomes a foundational principle of participation and coordination. Low-code components encapsulate not only functionality but also platform-specific design logic, with intrinsic properties such as dependency structures and configuration options. For example, a Mendix connector to Salesforce not only enables data integration but also enforces update rules and security standards defined by the platform, shaping how the component can be reused across applications. While SDKs and APIs also impose constraints—for instance, by exposing only certain functions or restricting access to system resources—LCDP components embed these rules more directly into the artifacts themselves. This means that constraints are carried within the component, influencing not just *what* can be built but *how* reuse and assembly occur in the visual development environment. In this way, they function simultaneously as enablers of generativity and instruments of control.

Platform governance

LCDP ecosystems share many governance characteristics with conventional digital platform ecosystems. As in traditional software platforms, roles are divided between platform providers (as orchestrators), third-party vendors (as contributors of infrastructure and reusable components), and developers (as both consumers and creators of application functionality) (Lusch & Nambisan, 2015; Tiwana et al., 2010). LCDP providers such as Mendix or OutSystems thus play roles analogous to Apple or Google, governing the technical architecture while setting participation rules that enable innovation (Ghazawneh & Henfridsson, 2013). Governance occurs through technical means (e.g., component standards, SDKs, APIs) and social mechanisms (e.g., documentation, training, developer forums) (Eaton et al., 2015). This is consistent with service-dominant logic views of platforms as co-creation environments (Lusch & Nambisan, 2015). Yet, LCDP ecosystem governance diverges from conventional platforms in several important ways, as explained below.

Citizen developer participation: Expanding the actor base Unlike traditional ecosystems, where developers are typically skilled software engineers, LCDPs also engage non-technical users ("citizen developers") (Elshan et al., 2025). These participants build applications for their own organizational needs and thereby blur the distinction between developer and end-user. This expanded user base requires LCDPs to embed more support into the development environment itself—such as guided workflows, templated solutions, and contextual help (Viljoen et al., 2024a)—to ensure safe and effective participation by non-experts.

Centralized orchestration: Curating and shaping ecosystems Like other platform orchestrators, LCDP owners govern the ecosystem by setting standards and defining participation rules. However, governance in LCDP ecosystems is more centralized and interventionist than in traditional open developer ecosystems, as LCDP owners go further by embedding these rules directly into the development environment itself. Thus, LCDP owners not only provide the infrastructure but also actively shape the ecosystem by producing and maintaining a substantial portion of reusable components themselves. Accordingly, as the ecosystem evolves, LCDP owners guide its development through infrastructure-level and component-level decisions, rather than relying on decentralized contributions or emergent

community consensus. The result is a curated platform environment in which participation is enabled but tightly structured to maintain coherence, quality, and strategic alignment.

Enterprise focus: Tailoring governance to organizational needs Whereas consumer-oriented platforms usually rely on market incentives or community-driven contributions, LCDPs must align ecosystem activity with enterprise requirements such as compliance, security, and interoperability with enterprise systems. Thus, governance mechanisms—such as role-based permissions, component certification processes, and partnership programs—are implemented not just to facilitate innovation but to ensure adherence to enterprise-grade standards. This results in governance regimes that extend beyond the relatively "light-touch" coordination seen in open-source or consumer app ecosystems.

Together, these dynamics culminate in what can be described as **curated enablement**: a governance logic where broad participation is fostered through controlled flexibility. As in conventional platforms, rules are embedded into technical artifacts—SDKs and APIs in mobile ecosystems already shape what developers can or cannot do. LCDPs, however, go further by integrating such rules directly into visual IDEs, reusable components, and guided workflows. In this way, *participation is not only bounded by external interfaces but also by the design of the development environment itself.* The result is a hybrid orchestration model that blends elements of open platform participation with the centralized control of enterprise software ecosystems.

Table 4 provides a synthesis of the conceptual similarities and distinctions of LCDP ecosystems.

Studying low-code development platform ecosystems: A research agenda

This Fundamentals paper showed how LCDP ecosystems can be conceptualized as a distinct class of digital platform ecosystems. Through our conceptualization, we identified several interesting research directions that researchers can undertake that could deepen our theoretical understanding of LCDP ecosystems. Accordingly, we propose a research agenda for studying LCDP ecosystems based on three structural elements of LCDP ecosystems. In Table 5, we present the research agenda and illustrative research questions.

Technical infrastructure of LCDP ecosystems

Design-oriented infrastructure The first avenue for future research is examining how infrastructural modularity and embedded design logic shape platform extensibility and generativity in LCDP ecosystems. While layered modular architectures are common in platform design, LCDPs conflate infrastructure and generativity through reusable components that embed platform-specific logic. These components not

Table 4 Conceptual similarities and distinctions of low-code development platform ecosystems

	Relevant concepts	Similarities with established concepts	Departure from established concepts
Technical infrastructure	• Platform-as-a-Service • Layered modular architecture • Containerization and serverless architectures	• LCDPs abstract low-level hardware concerns, similar to platforms-as-a-service • LCDPs consist of a layered modular architecture with a stable core of services and tools • LCDPs enable rapid, one-click deployment and scaling	• Reusable components are inputs for reuse rather than final outputs • LCDPs' IDEs are not just "toolkits," but a compositional space where generativity can be constrained • LCDPs are closed, proprietary environments
Boundary resources and software reuse	• Boundary resources • Packaged software reuse	• Components act as mediators between the platform and user, similar to SDKs • Low-code components facilitate reuse, similar to APIs and SDKs • Low-code marketplaces function like curated repositories, analogous to open-source software libraries	• Generative components are increasingly contributed by users, not just providers (endogenous creation) • Components embed governance into the artifact itself
Platform governance	• Platform orchestration • Service-dominant logic	• LCDP providers act as orchestrators, similar to those in traditional app stores (like Apple or Google) • Ecosystem governance takes place through both technical (component rules, SDKs) and social (training, documentation) mechanisms • Aligns with co-creation logics in platform literature	• Citizen developers blur developer/user boundaries, requiring embedded guidance • Governance is more centralized and prescriptive than in open-source or consumer platforms • Governance is more tailored to enterprise and organizational needs

Table 5 A research agenda for LCDP ecosystems

Ecosystem element	Research theme	Illustrative research questions
Technical infrastructure	***Design-oriented infrastructure***	• How do forms of infrastructural modularity shape the boundaries of platform extensibility? • In what ways does embedded infrastructure constrain or enable generativity?
	Infrastructure control vs. openness	• How can LCDP providers balance proprietary control with third-party extensibility? • What risks arise when platform owners rely on third-party infrastructure?
Reusable low-code components	***Reuse-centric modularity***	• How does the internal structure of low-code components affect their generativity across use cases and domains? • How can low-code reuse dimensions impact component uptake or combinatorial innovation?
	Component evolution and ecosystem value	• How does endogenous component creation by platform users influence ecosystem adoption and scalability? • What are the lifecycle patterns of reusable components in curated low-code marketplaces?
LCDP owner ecosystem governance	***Hybrid governance models***	• How does "curated enablement" differ from open-source or app store governance models? • On which levels can governance measures be integrated?
	Tensions between ecosystem actors	• What tensions occur between LCDP ecosystem actors? • How tensions between LCDP ecosystems be mitigated?
	Shift to citizen developers	• What onboarding, guidance, and support mechanisms are most effective for non-technical users? • How do platform governance needs shift as developer communities diversify?

only serve as outputs of development but also function as generative inputs. This creates a recursive dynamic where infrastructure enables reuse, and reuse redefines the generative potential of the infrastructure itself. While this aligns with recombinatorial innovation (Yoo et al., 2010), it challenges linear models of platform evolution. Future research could examine how boundary resources co-evolve with the modular architecture of the platform. Potential study designs include longitudinal case studies of LCDP providers (e.g., analyses of marketplaces' evolution) or large-scale surveys of developers assessing how infrastructure changes influence reuse practices.

Proprietary control As noted, unlike open developer ecosystems, LCDP providers maintain stronger architectural and infrastructural control and dictate both participation and reuse. However, while LCDP providers enact proprietary control, they themselves are dependent on proprietary third-party services (e.g., commercial APIs, AI engines, cloud platforms). These services are integral to LCDPs' functionalities and value propositions. Thus, these two sides of "proprietary logic"—internally imposed by platform owners and external dependence on third-party vendors—complicate platform governance and scalability. From the LCDP owner perspective, further research could investigate how reliance on external providers influences the competitive positioning of LCDPs. For example, when critical components such as cloud or generative AI connectors are primarily external, the LCDP's distinctive value proposition lies in orchestration, curation, and integration rather than in the components themselves. From the developer perspective, research could analyze how governance rules and external dependencies affect marketplace activity. A promising approach would be quantitative analysis of marketplace data (e.g., downloads, update cycles, component ratings) to uncover correlations between governance choices and developer participation.

Reusable low-code components

Reuse-centric modularity The second research direction centers on reusable components as the defining boundary resources in LCDP ecosystems. This paper highlighted five reuse dimensions—two backend and three front-end—that shape how components function as modular building blocks in LCDP ecosystems. This reuse-centric modularity is not inherent but actively designed and reflects provider choices about how functionality, access, and scope are packaged. These structural decisions also directly impact a component's generativity, meaning its ability to be repurposed across use cases and domains. For example, highly specific components may lack flexibility, while overly generic ones may demand extensive customization. Understanding how

these structural design choices affect adoption and recombination in practice could help LCDP providers develop components that balance specificity, configurability, and reuse potential. Moreover, exploring how the combination or permutation of reuse dimensions affects marketplace performance (e.g., component downloads, update frequency, etc.) can offer valuable insights for component creators to develop and publish useful components.

Component evolution and ecosystem value As component reuse becomes increasingly endogenous, these components act not only as technical enablers but also as evolving resources that structure ecosystem growth. Future research can analyze how reuse-centric modularity—including domain specificity, internal configurability, and dependency patterns—shapes the generativity and adoption of components. Lifecycle patterns of components in curated marketplaces—including how they are iterated, maintained, or deprecated—could further explain ecosystem dynamics. Potential approaches include longitudinal case studies of component lifecycles (e.g., how Salesforce or SAP connectors evolve across platform versions) and quantitative mining of marketplace data.

Platform owner ecosystem governance

Hybrid governance models The third research direction addresses the hybrid governance logic that distinguishes LCDP ecosystems from more open, decentralized software ecosystems. As discussed, LCDP providers like Mendix and OutSystems operate under a model of *curated enablement* and thereby combine broad accessibility with centralized control. Future work could examine how this governance logic compares to established forms such as open-source communities or mobile app stores. This can be done by analyzing how control is exercised across technical, social, and organizational layers. For example, governance may be embedded directly into development workflows (e.g., through templates or role-based access), instantiated in the component architecture (e.g., standardized configurations), or enforced through platform policies. Comparative case studies of platforms like Mendix, OutSystems, and Android could map governance interventions at each layer. Experimental studies that embed role-based restrictions into IDE prototypes could assess their effect on citizen developer participation and application quality.

Shift to citizen developers As LCDPs are unique in their inclusion of non-expert developers (citizen developers), this must also be taken into account in future research. While governance for citizen development has been investigated from an organizational perspective (Carroll et al., 2024)—typically focusing on internal policies, training programs, and IT oversight—the interplay from an ecosystem perspective remains unexplored. For example, as LCDP ecosystems diversify, platform governance may not solely rely on procedural oversight or organizational support, but must also be instantiated into the development environment through embedded logic and interface design. Further research could analyze how governance mechanisms accommodate both professional and citizen developers, and how they evolve alongside shifts in the developer base. Comparative case studies of enterprises adopting citizen development at scale could reveal ecosystem-level effects. Experimental studies that vary interface design features (e.g., embedded guidance, error prevention mechanisms) could measure their impact on citizen developer effectiveness and participation.

Actor tensions in LCDP ecosystems A further avenue for research lies in exploring tensions between ecosystem actors. For example, while boundary resources in LCDPs are increasingly created by external actors, LCDP owners simultaneously exert centralized governance by curating marketplaces, enforcing standards, and embedding constraints into tools. This duality creates paradoxical tensions (Smith & Lewis, 2011): ecosystem growth depends on third-party and even citizen developer contributions, yet platform owners must retain control to ensure security, compatibility, and enterprise reliability. Such tensions resemble those observed in generative AI ecosystems (Viljoen et al., 2024b), where providers depend on external data and models but also seek to control proprietary assets. Investigating how these tensions play out could draw on longitudinal case studies of platforms like Mendix Marketplace or Microsoft Power Platform, or configurational approaches such as qualitative comparative analysis to identify patterns in how LCDP owners and contributors jointly manage these tensions.

Conclusion

LCDPs are reshaping the logic of software innovation by embedding reuse, governance, and visual design into tightly orchestrated platform ecosystems. This Fundamentals paper conceptualized LCDPs as a distinct class of digital platform ecosystem, characterized by embedded technical infrastructure, reusable low-code components as boundary resources, and curated platform ecosystem governance. We showed how these elements align with—but also depart from—established notions of digital platform ecosystems, boundary resources, packaged software reuse, and platform governance. In sum, LCDPs replace traditional notions of openness with curated enablement and recast reuse not as a downstream practice but as a foundational "upstream" design logic.

Accordingly, as LCDP ecosystems challenge assumptions about how platforms scale, who contributes, and how innovation unfolds, we proposed a forward-looking research agenda centered around the unique dynamics of infrastructure, component design, and platform ecosystem governance in low-code contexts. As LCDPs continue to grow in enterprise environments and beyond, they present an important opportunity for platform scholars to revisit foundational concepts such as modularity, generativity, and governance. We thus encourage researchers to treat LCDPs not as marginal tools for end-user development, but as rich, theory-relevant settings offering interesting new phenomena.

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/s12525-025-00848-x.

Acknowledgement We thank Nhi Ngo for her early exploratory analysis of the Mendix platform and marketplace.

Funding Open Access funding enabled and organized by Projekt DEAL.

Declarations

Competing interests The authors declare that they have no competing interests.

References

Al Alamin, M. A., Uddin, G., Malakar, S., Afroz, S., Haider, T., & Iqbal, A. (2023). Developer discussion topics on the adoption and barriers of low code software development platforms. *Empirical Software Engineering, 28*, Article 4. https://doi.org/10.1007/s10664-022-10244-0

Baldwin, C. Y., & Woodard, C. J. (2009). The architecture of platforms: A unified view. *Platforms, markets and innovation* (pp. 19–44). https://doi.org/10.2139/ssrn.1265155

Bock, A. C., & Frank, U. (2021). Low-code platform. *Business & Information Systems Engineering, 63*(6), 733–740. https://doi.org/10.1007/s12599-021-00726-8

Bucaioni, A., Cicchetti, A., & Ciccozzi, F. (2022). "Modelling in low-code development: A multi-vocal systematic review." *Software and Systems Modeling*. https://doi.org/10.1007/s10270-021-00964-0

Carroll, N., Holmström, J., & Matook, S. (2024). Special issue editorial: Transforming business with low-code and no-code. *MIS Quarterly Executive, 23*(3), v–xiii.

Cennamo, C., & Santaló, J. (2019). Generativity tension and value creation in platform ecosystems. *Organization Science, 30*(3), 617–641. https://doi.org/10.1287/orsc.2018.1270

Chen, L., Tong, T. W., Tang, S., & Han, N. (2022). Governance and design of digital platforms: A review and future research directions on a meta-organization. *Journal of Management, 48*(1), 147–184. https://doi.org/10.1177/01492063211045023

Cusumano, M., Gawer, A., & Yoffie, D. B. (2019). *The business of platforms: Strategy in the age of digital competition, innovation, and power*. Harper Business.

De Reuver, M., Sørensen, C., & Basole, R. C. (2018). The digital platform: A research agenda. *Journal of Information Technology, 33*(2), 124–135. https://doi.org/10.1057/s41265-016-0033-3

Di Ruscio, D., Kolovos, D., De Lara, J., Pierantonio, A., Tisi, M., & Wimmer, M. (2022). Low-code development and model-driven engineering: Two sides of the same coin? *Software and Systems Modeling, 21*, 437–446. https://doi.org/10.1007/s10270-021-00970-2

Eaton, B., Elaluf-Calderwood, S., Sørensen, C., & Yoo, Y. (2015). Distributed tuning of boundary resources. *MIS Quarterly* 39(1), 217–244. https://doi.org/10.25300/MISQ/2015/39.1.10

Eggers, J., Wewerka, J., Viljoen, A., & Krcmar, H. (2023). Supporting subject matter experts as developers: Towards a framework for decentralized robotic process automation development. In *Proceedings of the 56th Hawaii International Conference on System Sciences*. https://doi.org/10.24251/HICSS.2023.663

Elshan, E., Binzer, B., & Winkler, T. J. (2025). From software users to software creators: An exploration of the core characteristics of the citizen developer role and the related re- and upskilling programs. *Business & Information Systems Engineering, 67*(1), 31–53. https://doi.org/10.1007/s12599-024-00915-1

Elshan, E., Siemon, D., Bruhin, O., Kedziora, D., & Schmidt, N. (2024). Unveiling challenges and opportunities in low code development platforms: A stackoverflow analysis. In *Proceedings of the 57th Hawaii International Conference on System Sciences*. https://doi.org/10.24251/HICSS.2024.872

Evans, P. C., & Gawer, A. (2016). *The rise of the platform enterprise: A global survey* (The Emerging Platform Economy Series No. 1). The Center for Global Enterprise.

Fürstenau, D., Baiyere, A., Schewina, K., Schulte-Althoff, M., & Rothe, H. (2023). Extended generativity theory on digital platforms. *Information Systems Research, 34*(4), 1686–1710. https://doi.org/10.1287/isre.2023.1209

Gartner. (2024). *Information technology glossary: Citizen developer*. Retrieved February 27, 2024, from https://www.gartner.com/en/information-technology/glossary/citizen-developer

Ghazawneh, A., & Henfridsson, O. (2013). Balancing platform control and external contribution in third-party development: The boundary resources model. *Information Systems Journal, 23*(2), 173–192. https://doi.org/10.1111/j.1365-2575.2012.00406.x

Hein, A., Schreieck, M., Riasanow, T., Setzke, D. S., Wiesche, M., Böhm, M., & Krcmar, H. (2020). Digital platform ecosystems. *Electronic Markets, 30*(1), 87–98. https://doi.org/10.1007/s12525-019-00377-4

Hein, A., Weking, J., Schreieck, M., Wiesche, M., Böhm, M., & Krcmar, H. (2019). Value co-creation practices in business-to-business platform ecosystems. *Electronic Markets, 29*(3), 503–518. https://doi.org/10.1007/s12525-019-00337-y

Henfridsson, O., & Bygstad, B. (2013). The generative mechanisms of digital infrastructure evolution. *MIS Quarterly* (pp. 907–931). https://doi.org/10.25300/MISQ/2013/37.3.11

KPMG. (2024). *Low-code adoption as a driver of digital transformation*. KPMG Report February, KPMG.

Lusch, R. F., & Nambisan, S. (2015). Service innovation: A service-dominant logic perspective. *MIS Quarterly* 39(1), 155–176. https://doi.org/10.25300/MISQ/2015/39.1.07

Mendix. (2024). *AWS*. Retrieved February 19, 2025, from https://docs.mendix.com/partners/aws/

Mendix. (2025a). *Blank web app*. Retrieved July 4, 2025, from https://marketplace.mendix.com/link/component/51830

Mendix. (2025b). *Deployment*. Retrieved July 3, 2025, from https://www.mendix.com/evaluation-guide/deployment/

Mendix. (2025c). *Explore Mendix marketplace*. Retrieved February 14, 2025, from https://marketplace.mendix.com/

Mendix. (2025d). *The Mendix Atlas UI framework*. Retrieved July 1, 2025, from https://www.mendix.com/atlas/

Microsoft. (2020). *Coca-Cola Bottling Company United dispenses streamlined order management with RPA in Microsoft Power Automate*. Retrieved June 6, 2024, from https://customers.microsoft.com/en-us/story/845187-coca-cola-bottling-company-united-consumer-goods-power-automate

OutSystems. (2024). *The complete guide to creating components*. Retrieved February 19, 2025, from https://success.outsystems.com/documentation/best_practices/development/the_complete_guide_to_creating_components/

OutSystems. (2025a). *Forge*. Retrieved February 14, 2025, from https://www.outsystems.com/forge/list

OutSystems. (2025b). *Integrate outsystems with external version controls*. Retrieved July 3, 2025, from https://success.outsystems.com/documentation/how_to_guides/devops/integrate_outsystems_with_external_version_controls/

Prinz, N., Rentrop, C., & Huber, M. (2021). Low-code development platforms–A literature review. In *Proceedings of the 27th Americas Conference on Information Systems*. https://aisel.aisnet.org/amcis2021/adv_info_systems_general_track/adv_info_systems_general_track/2

Richardson, C., & Rymer, J. R. (2014). *New development platforms emerge for customer-facing applications*. Forrester, Cambridge, MA.

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

Scharpf, D., Viljoen, A., Hein, A., & Krcmar, H. (2024). Business unit development: Benefits and challenges for employees. *HMD Praxis der Wirtschaftsinformatik, 61*, 1159–1179. https://doi.org/10.1365/s40702-024-01094-z

ServiceNow. (2024). *Novant health unlocks the limitless potential of people*. Retrieved June 6, 2024, from https://www.servicenow.com/customers/novant-health-citizen-development.html

Smith, W., & Lewis, M. (2011). Toward a theory of paradox: A dynamic equilibrium model of organizing. *Academy Of Management Review, 36*(2), 381–403. https://doi.org/10.5465/amr.2009.0223

Sojer, M., & Henkel, J. (2010). Code reuse in open source software development: Quantitative evidence, drivers, and impediments. *Journal of the Association for Information Systems, 11*(12), 868–901. https://doi.org/10.17705/1jais.00248

Star, S. L. (2010). This is not a boundary object: Reflections on the origin of a concept. *Science, Technology, & Human Values, 35*(5), 601–617. https://doi.org/10.1177/0162243910377624

Staub, N., Haki, K., Aier, S., & Winter, R. (2023). Governance mechanisms in digital platform ecosystems: Addressing the generativity-control tension. *Communications of the Association for Information Systems* 52(1), 906–939. https://doi.org/10.17705/1CAIS.05137

Tiwana, A., Konsynski, B., & Bush, A. A. (2010). Platform evolution: Coevolution of platform architecture, governance, and environmental dynamics. *Information Systems Research, 21*(4), 675–687. https://doi.org/10.1287/isre.1100.0323

Vial, G. (2023). A complex adaptive systems perspective of software reuse in the digital age: An agenda for is research. *Information Systems Research, 34*(4), 1728–1743. https://doi.org/10.1287/isre.2023.1200

Viljoen, A., Radić, M., Hein, A., Nguyen, J., & Krcmar, H. (2024a). Governing citizen development to address low-code platform challenges. *MIS Quarterly Executive*, *23*(3), 305–324. https://aisel.aisnet.org/misqe/vol23/iss3/6

Viljoen, A., Hein, A., Constantinides, P., & Krcmar, H. (2024b). Leveraging generative AI in enterprise contexts: Towards a paradox theory and organizational boundary work approach. In *Proceedings of the 45th International Conference on Information Systems*. https://aisel.aisnet.org/icis2024/ent_system/ent_system/4/

Viljoen, A., Stelzl, B., Yang, M., Nguyen, J., Hein, A., Elshan, E., & Kcrmar, H. (2025). Navigating flexibility and standardization in low-code/no-code development. *Information Systems Journal*. https://doi.org/10.1111/isj.70001

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

Von Hippel, E., & Katz, R. (2002). Shifting innovation to users via toolkits. *Management Science, 48*(7), 821–833. https://doi.org/10.1287/mnsc.48.7.821.2817

Wareham, J., Fox, P. B., & Cano Giner, J. L. (2014). Technology ecosystem governance. *Organization Science, 25*(4), 1195–1215. https://doi.org/10.2139/ssrn.2201688

Weber, M., Hein, A., Weking, J., & Krcmar, H. (2024). Orchestration logics for artificial intelligence platforms: From raw data to industry-specific applications. *Information Systems Journal, 35*(3), 1015–1043. https://doi.org/10.1111/isj.12567

Yoo, Y., Henfridsson, O., & Lyytinen, K. (2010). The new organizing logic of digital innovation: An agenda for information systems research. *Information Systems Research, 21*(4), 724–735. https://doi.org/10.1287/isre.1100.0322