API Management Maturity of Low-Code Development Platforms[⋆]

Michiel Overeem ✉[1] [0000-0003-4807-4124], Slinger Jansen[2,3] [0000-0003-3752-2868], and Max Mathijssen[2]

[1] AFAS Software, Inspiratielaan 1, Leusden, The Netherlands
michiel.overeem@afas.nl
[2] Utrecht University, Princetonplein 5, Utrecht, The Netherlands
{slinger.jansen,m.mathijssen}@uu.nl
[3] Visiting Scientist, School of Engineering Science, LUT University, Finland

Abstract. Low-code development platforms are environments that enable *citizen developers* without software engineering knowledge to create software products. These software products range from small business applications to large business platforms, around which software ecosystems increasingly form. In these software ecosystems, different organizations want to extend the created software products with services and software, with the goal of creating active enterprise networks that create value collaboratively. Well designed and maintained application programming interfaces are crucial for these organizations.
In this paper we evaluate the application programming interface management maturity of four low-code development platforms. We show that these platform providers are not yet concerned with helping their customers build software ecosystems around the software platforms that citizen developers create. Furthermore, we identify the *software engineering research challenges* that these platform providers face. For instance, low-code development platforms should create abstractions that let *citizen developers* design, develop, and manage application programming interfaces. If low-code development platform providers follow our advice and act on it, they will become able to provide customers with complete ecosystem-enabled platforms instead of providing only simple throwaway business applications.

Keywords: Low-Code · Model-Driven Development · API Management · Citizen Software Development

1 Introduction

Increasingly, traditional Software Producing Organizations, i.e., organizations whose main activities include the production of software such as software vendors and open source organizations, are discovering platforms as a vehicle to increase the value of their software for their customers through collaboration with

⋆ This research was supported by the NWO AMUSE project (628.006.001): a collaboration between Vrije Universiteit Amsterdam, Utrecht University, and AFAS Software in the Netherlands. See amuse-project.org for more information . The final authenticated version is available online at https://doi.org/10.1007/978-3-030-79186-5_25.

third parties. This transformation from product towards a platform is called 'platformisation' [14]. Platforms are a vehicle for software ecosystems and are defined as a set of organizations collaboratively serving a market for software and services [9]. These ecosystems form around software platforms, which in turn are managed by software platform orchestrators.

We find that not all software products can easily transform into software platforms. One particular category of software products are products that are created using no-code/low-code development platforms (LCDPs). LCDPs apply model-driven development to raise the abstraction level of software development, increasing productivity and decreasing complexity as a result [3]. They target a wide variety of users, from professional software developers to non-technical business experts [13]. The latter category is commonly referred to as the 'citizen developer': people without software development education who nonetheless build software applications. Customers of these LCDPs utilize these platforms to increase their agility, it enables them to develop business applications more efficiently without suffering from the lack of professionally trained software developers. While LCDPs are traditionally used to create agile business software applications, they are increasingly becoming part of the core IT landscape [17]. As long as the LCDPs prove their value, customer companies will utilize them in more and more diverse projects. The customer companies will be interested in evolving their application into a platform to enable complementors, which are organizations that build application that extend the core system, to create more value [9].

Evolving into an ecosystem is done by offering parts of the developed application through Application Programming Interfaces (APIs). Research shows that concerns such as stability, security, and scalability in ecosystems are related to API management capabilities [1]. API management is the activity that enables organizations to design, publish, and deploy their APIs for (external) developers to consume. This is one of the enabling practices for the creation of an ecosystem. Traditionally the activities within API management are executed by technical staff members. The strength of LCDPs, however, is that activities that used to be executed by highly technical staff, such as software engineers, are now executed by citizen developers. To support these API management activities LCDPs have to provide the means to the citizen developers to integrate applications developed on the LCDP with other applications.

We believe that LCDP providers must mature their API management capabilities to remain relevant for citizen developers. After all, software ecosystems are increasingly seen as the way to remain strategically relevant in the software industry [8]. In this paper, we distinguish between two software ecosystems that form around LCDPs. First, the LCDP ecosystem is the ecosystem that forms around the LCDP and the provider of the platform. The second is the LCDP Application Ecosystem, which forms around the application that is created by one of the customers of the LCDP. Our contribution is an evaluation of the maturity of API management capabilities support among LCDPs, along with a set of challenges that we identify. We evaluate the capabilities of four LCDPs through descriptive case studies. In these case studies we measure the maturity of their API management capabilities through a Focus Area Maturity Model (FAMM). A FAMM is a model that groups capabilities and practices in focus areas and aligns them along maturity levels, which can be used to evaluate organizational practices around particular focus areas [2, 8].

Section 2 describes our research method and gives a short description of the four LCDP providers investigated in our evaluating case studies. The API management FAMM (API-m-FAMM) is described in Section 3. In Section 4 we evaluate the LCDPs to uncover their level of support for a citizen developer in API management activities. The results of the case studies are analyzed in Section 5. We contribute a set of engineering research challenges for software engineering researchers in Section 6. Threats to validity are discussed in Section 7. In Section 8 we observe, through the four case studies, that some LCDPs are slowly maturing their API management capabilities while others do not act on these opportunities. Furthermore, we observe that the API-m-FAMM, while mostly aimed at organizations with a proprietary API infrastructure, is also applicable to LCDPs. Finally, we conclude that these platform providers need to increase the level of abstraction of the technical complexity of managing APIs, without losing the strategic strengths of it.

2 Research Method

Our research focuses on the question: *How mature are the API management capabilities that LCDPs offer?* We use an established framework for evaluation of API management maturity, to evaluate how well applications, created with four LCDPs, enable API capabilities for the LCDP customer.

In our research we apply a FAMM to measure the maturity of API management in LCDPs. The API-m-FAMM [11] captures API management in 81 practices, grouped in 20 capabilities, which are in turn assigned to six focus areas. The model is intended for organizations that develop their own proprietary API infrastructure. However, in our case studies we evaluate the LCDPs, to uncover how they support the API management activities of their customers. We apply a maturity model to measure the current state and provide a roadmap to advance this state, similar to [5]. Please note that more details are provided on the API-m-FAMM in the next Section.

The evaluations are performed in four descriptive case studies, which were conducted with the ACM SIG Empirical Research standard in mind [16]. To be able to extrapolate our findings we selected platforms that represent the current state of LCDPs. Our selection was made based on the Quadrant for Enterprise LCDPs of the advisory company Gartner [19]. This report surveys 18 platforms and categorizes them into leaders, visionaries, challengers, and niche players. We selected two leaders and two visionaries that were willing to cooperate in our evaluation. From Gartner's report we can conclude that leaders and visionaries will show the state of the art in Enterprise LCDPs and represent the most advanced platforms.

The case studies were conducted with the following steps. First, public sources such as product documentation and company blogs were studied to create a first impression of the API management capabilities of the LCDP. Next, an interview with a company expert (either a product manager, architect, or chief technology officer) was conducted. During the interview the API-m-FAMM [11] was described to the interviewee, including all of the practices and their terminology. Together with the interviewee the API-m-FAMM was used to asses the LCDP. Finally, the interviewees discussed their LCDP with respect to API management, and their opinion on creating platforms on top of the LCDP. Subsequently the interviews were processed and analyzed. Based on this analysis and together with the company documentation the evaluation of the LCDP using the API-m-FAMM was completed. Afterwards the evaluation was shared with the interviewee to correct mistakes and oversights. Finally, our general findings were discussed with the interviewees, to establish how they perceive the role of APIs in their generated applications and whether they equally acknowledge the trend of ‘ecosystemification’.

We shortly describe the case study organizations here. *Mendix* is a worldwide operating low-code platform provider, founded in the early 2000s. The company employs 1,000 employees, serving thousands of customer companies and an ecosystem of almost 150,000 developers. *OutSystems*, as the second biggest and oldest provider, has been active for almost 20 years. With well over 1,000 employees worldwide, they serve a large range of companies. The LCDP originated as a rapid application development platform. *Betty Blocks*, founded almost 10 years ago, employs around 200 people. Operating worldwide, they serve customers in all business domains. This LCDP has a strong focus on the citizen developer in enterprises, as reflected in their vision: ‘anyone should be able to build an application.’ *Pega* is the oldest (40 years) and biggest (6,000 employees) company of the four. Its LCDP evolved from a business process modelling suite.

3 Introduction of the API-m-FAMM

The goal of the API-m-FAMM[4] is to support organizations that expose their APIs to third-party developers in their API management activities. Using the API-m-FAMM, organizations may evaluate, improve upon and assess the degree of maturity their API management processes have.

A focus area maturity model [18] consists of focus areas, and an area consists of capabilities, which are defined as *the ability to achieve a certain goal related to API management, through the execution of two or more interrelated practices.* A practice in turn is defined as *an activity that has the express goal to improve, encourage and manage the usage of APIs.* The API management maturity model is created following the steps described by [18] and [2]. The scope, design, and populate phase are based on a systematic literature review [10]. The model was further refined through two rounds of interviews with experts: eleven interviews

[4] A detailed description and the source data are published [11]. The model is also available on the `https://MaturityModels.org` web site.

in the first round and three interviews in the second round. Finally the model was used to asses five different software products.

The API-m-FAMM consists of six focus areas that we briefly summarize here:

- **Lifecycle Management**: An API undergoes several stages over the course of its lifetime [12]. Version management is particularly challenging: complementors in the ecosystem benefit from stable APIs, but at the same time demand new functionality to further their own product.
- **Security**: APIs provide access to valuable and protected data and assets. Therefore, mature APIs implement the latest security standards, such as the *OAuth 2.0* authorization protocol, and protection against threats such as Denial of Service attacks.
- **Performance**: APIs deliver data and services to complementors in the ecosystem. This increases the demand on APIs to perform well under load: the application itself as well as the complementors are negatively effected by a decrease in performance.
- **Observability**: An organization benefits from insight into the API's usage. Through various monitoring techniques, the organization is able to collect metrics which can shed light on the API's health and performance, as well as its usage by complementors. A performant and healthy API is crucial, because an interrupted service of the APIs will also most likely interrupt the complementors application.
- **Community**: It is desirable for organizations to foster, engage, and support the community that exists around the API. This entails offering developers the ability to register for API access and offering them access to test environments, code samples, and documentation.
- **Commercial**: Exposing and consuming APIs can have a commercial aspect tied to it [4]. On one hand, APIs can require a subscription fee from the complementors, on the other hand complementors might demand Service Level Agreements from the provider.

These focus areas are composed of 20 capabilities, which in turn comprise 81 practices. Within their corresponding capabilities, which may be regarded as sub-topics, practices are ranked based on the perceived complexity of their implementation. In order to verify whether an organization has implemented a practice, a set of conditions for implementation has been defined for each practice. By examining the fulfillment of the aforementioned implementation conditions, it may be determined whether an organization has implemented a practice. When this is done for each practice a capability consists of, an organization's *maturity level* for that capability may be determined.

We provide a description of the practice *Implement Multiple API Versioning Strategy* here, to clarify how the practices are evaluated. The description of this practice is *"The organization has a versioning strategy in place which entails the process of versioning from one API to a newer version. In order to do so, the organization must be able to maintain multiple versions of (one of) their API(s) for a period of time. Possible strategies include URI/URL Versioning (possibly* *in combination with adherence to the Semantic Versioning specification), Query Parameter versioning, (Custom) Header versioning, Accept Header versioning or Content Negotiation."* Each practice has an *Implemented when* text, that describes one or more conditions to evaluate whether a practice has been implemented or not. In this case the condition is self-explanatory: "The organization utilizes one of the following versioning strategies: URI/URL Versioning, Query Parameter versioning, (Custom) Header versioning, Accept Header versioning or Content Negotiation.". For the LCDPs, we discussed whether it is possible to maintain different versions of the API, or whether an API always co-evolves with the model and does not have any kind of evolution mechanisms implemented to enable different API versions.

4 Case Studies

This Section describes the four evaluations of the LCDPs that were done with the API-m-FAMM. This assessment was done based on the available platform documentation and the interview. The interviewees were able to point out mistakes in the evaluation, comments were incorporated accordingly. First we describe the LCDPs in general, then we discuss the six focus areas and how the LCDPs support these.

Mendix - The road map of *Mendix* shows a focus towards enabling citizen developers to create increasingly complex applications, which is motivated by two developments. First of all, applications developed on the LCDP are growing and becoming increasingly complex. Second, an increase in demand from citizen developers to build integrated applications independent from professional developers is observed. Strong API management capabilities enable customers to split their large applications into smaller integrated applications. The envisioned central API catalog will bring together applications within an enterprise, enabling citizen developers to develop integrated solutions.

OutSystems - The focus of *OutSystems* is ensuring that the co-development between the citizen developer and the professional developer is made as efficient as possible. Their vision is 'fast and agile development of enterprise applications'. API management is not hidden behind abstractions, but rather placed in the hands of professional developers. The gap between technical API management and the citizen developers is not actively bridged, instead the co-development between citizen developers and professional developers is promoted.

Betty Blocks - This LCDP is focused on consuming APIs, instead of publishing them. *Betty Blocks* states that their LCDP is not used to develop core systems, but to develop supporting applications. Applications mostly complement existing systems. The runtime of the LCDP consists of a web-based server and browser-based client application. Developers do not have to explicitly design APIs, as every application feature is an API by default through this architecture.

Pega - Ease of change and rapid application development by collaborating departments in enterprises are the focus of *Pega*. The developer tool of the LCDP supports multiple personas, both the citizen developer as well as the professional developer. *Pega* supports a myriad of integration options, among REST or SOAP APIs it also supports integration through database connection or e-mail. Despite plentiful of options to integrate with other applications, *Pega* does not observe a large portion of their customers using these capabilities to integrate with complementors outside of the organization.

Through the API-m-FAMM evaluation we measure the state of API management support that the LCDPs offer. Considering that the API-m-FAMM is targeted towards organizations that expose their APIs to third-party developers, the evaluation of LCDPs differs from this original intention. A LCDP is both an application (run-time) platform and a development platform. API management practices can be implemented in different ways in a LCDP:

- A practice can be **statically implemented** by the LCDP, meaning that the customer cannot influence it (an example is *Load balancing*).
- **Variable** implemented practices are those practices that can be influenced by citizen developers or professional developers, such as *Multiple API Versions Strategy*.
- Some practices can only be implemented by using products from **third-party** vendors. Example is the *Adopt Subscription-Based Monetization Model* practice.
- The LCDP is a development environment and in that capacity the LCDP can be used to implement a number of API management practices. Examples of these **build-your-own** practices are *Community Forum* and *Broadcast API Status*.

The last two categories, third-party and build-your-own, result in more work for the customers of the LCDP. They become responsible for developing and maintaining these specific practices, while statically and variable implemented practices do not have these liabilities. Therefore, we evaluate the four LCDPs by scoring the practices in two categories: *supported* (by the LCDP) and *custom* (developed) practices. The first category consists of all statically and variable implemented practices, third-party and build-your-own practices are grouped in the second category. Table 1 shows the results per API-m-FAMM capability[5]. Every score consists of two numbers: first the number of practices that are supported by the LCDP, then the number of practices that need to be custom implemented.

Focus Area: Lifecycle Management - Generally speaking, the LCDPs support both the consumption and publication of modern APIs. Standard protocols such as SOAP and REST are supported by all LCDPs. *Mendix*, *OutSystems*, and *Pega* also support the API protocol OData. The decision of *Betty Blocks* to create APIs automatically shows a strong opinion on APIs. Some practices are not implemented, because their choice for GraphQL based APIs enforces APIs

[5] The detailed evaluations are available through `http://dx.doi.org/10.17632/wdtg5ytdpf.1`

Table 1. Evaluation of the API management maturity of the four LCDPs according to the API-m-FAMM: *Mendix* (M), *OutSystems* (O), *Betty Blocks* (B) and *Pega* (P). For every API-m-FAMM capability we show the total number of practices, and the LCDP evaluation. The two numbers per LCDP stand for practices *supported* by the LCDP and practices that need to be *custom* developed respectively.

	Focus Area	**M**	**O**	**B**	**P**
1	**Lifecycle Management**	**7/5**	**8/4**	**6/4**	**8/4**
1.1	Version Management (*4 practices*)	2/2	3/1	2/1	3/1
1.2	Decoupling API & Application (*4 practices*)	4/0	4/0	3/0	4/0
1.3	Update Notification (*4 practices*)	1/3	1/3	1/3	1/3
2	**Security**	**12/4**	**12/4**	**10/4**	**12/4**
2.1	Authentication (*3 practices*)	3/0	3/0	2/0	3/0
2.2	Authorization (*4 practices*)	4/0	4/0	4/0	4/0
2.3	Threat Detection & Protection (*6 practices*)	3/3	3/3	2/3	3/3
2.4	Encryption (*3 practices*)	2/1	2/1	2/1	2/1
3	**Performance**	**6/5**	**6/5**	**8/2**	**7/4**
3.1	Resource Management (*4 practices*)	3/1	3/1	3/1	3/1
3.2	Traffic Management (*7 practices*)	3/4	3/4	5/1	4/3
4	**Observability**	**5/7**	**5/7**	**5/7**	**5/7**
4.1	Monitoring (*3 practices*)	0/3	0/3	0/3	0/3
4.2	Logging (*4 practices*)	4/0	4/0	4/0	4/0
4.3	Analytics (*5 practices*)	1/4	1/4	1/4	1/4
5	**Community**	**10/8**	**9/9**	**10/8**	**8/10**
5.1	Developer Onboarding (*4 practices*)	4/0	4/0	4/0	4/0
5.2	Support (*3 practices*)	1/2	1/2	1/2	0/3
5.3	Documentation (*3 practices*)	2/1	2/1	2/1	2/1
5.4	Community Engagement (*5 practices*)	0/5	0/5	0/5	0/5
5.5	Portfolio Management (*3 practices*)	3/0	2/1	3/0	2/1
6	**Commercial**	**2/10**	**2/10**	**2/10**	**2/10**
6.1	Service-Level Agreements (*4 practices*)	2/2	2/2	2/2	2/2
6.2	Monetization Strategy (*4 practices*)	0/4	0/4	0/4	0/4
6.3	Account Management (*4 practices*)	0/4	0/4	0/4	0/4
	Total	**42/39**	**42/39**	**41/35**	**42/39**

without versions. Their LCDP customers cannot implement a versioning strategy, considering that API consumers always use the latest version, and customers need to take care of backwards compatibility. The other capabilities *Decoupling API & Application* and *Update Notification* show great resemblance between the four LCDPs. In the first capability all practices are supported by the LCDPs, while customers are expected to *custom* implement most practices in the second capability.

Focus Area: Security - In the area Security there is almost no differentiation between the LCDPs. All of the LCDPs follow modern security standards such as *Implement Transport Layer Encryption*, *Implement Authentication Protocol*, and *Implement Access Protocol.* The platforms support their customers in most practices.

Only the capability *Threat Detection & Protection* has a number of advanced practices that need to be implemented by the LCDP customers: *Security Breach Protocol*, *Conduct Security Review*, and *Implement Zero Trust Network Access*. While the providers have implemented these practices for their own hosted services, customers are responsible for their own protocols and are thus required to implement these practices as well.

Focus Area: Performance - Again the LCDPs are similar in their support of the practices in this area. In the *Resource Management* capability we observe that the LCDPs implement most of the practices. *Load Balancing*, *Scaling*, and *Failover* are all supported by the providers. Advanced practices in *Traffic Management* are mostly left to be *custom* implemented by the LCDP customers. The LCDP customers are required to configure third-party applications that implement practices such as *Manage Quota* and *Prioritize Traffic*. Implementing these practices requires expertise of professional developers and thus extra investment from the customers.

Focus Area: Observability - In this area there is no difference between the LCDPs. The practices in the capability *Logging* are supported by all four platforms. Monitoring the health, performance and resource consumption of APIs is left to the customers. *Custom Analysis Reports*, *Status Broadcasting*, and *Alerts* are also left to be custom implemented.

Focus Area: Community - Practices from the area *Community* that focus on technical capabilities, such as *Software Development Kit Support* and *API Catalog* are supported by the LCDPs. All of the LCDPs implement the API specification language OpenAPI, which supports practices such as *Use Standard for Reference Documentation* and *Provide SDK Support*. Less technical practices, such as *Social Media Presence* and *Communication Channel* are left to the customers to implement. Some of these practices, such as *Community Form*, can be built on top of the LCDP.

Focus Area: Commercial - The area *Commercial* is underdeveloped in all four LCDPs. Developers are not able to monetize the APIs developed on top of the LCDP, neither are they able to construct custom Service Level Agreements towards their API consumers. In order for customers to implement these practices they are required to integrate with third-party API management solutions.

5 Analysis of the Results

The four LCDPs under study show a great resemblance when evaluating their API management maturity. The fact that they are all either leaders or visionaries in the Quadrant for Enterprise LCDPs makes this no surprise, considering that they are ranked similarly. However, what is surprising is the general lack of support for advanced API management practices.

Mendix supports 42 practices, leaving 39 practices to be implemented by their consumers, making it an almost 50-50 split. The roadmap, as discussed during the interview, shows a focus on supporting their customers in the API management activities. This support is aimed at making it easier for citizen developers to build and publish APIs of higher quality. This roadmap has a focus on the practices in the *Community* practice. The area *Commercial* is not on the roadmap, making it harder for customers to monetize their APIs.

OutSystems supports mostly the same practices as *Mendix*, but made it clear during the interview that their ambitions differ. They have a strong focus on creating a platform that can enable the development of core enterprise systems by teams consisting of both citizen and professional developers. In this vision, there is no need to support all practices, because the provider recognizes that their customers already have several enterprise API platform solutions in place. Therefore, although there is a mature platform, many of the API management practices are left to their customer to implement themselves.

Within *Betty Blocks* (supporting 41 practices), publishing APIs is possible, but the capabilities are not mature enough to build a platform. In agreement with their vision, the LCDP can be used to complement other applications, but is less usable to create core systems. This is caused by two main reasons. First of all, their opinionated implementation of API versioning through GraphQL limits customers in how they want to expose APIs to their complementors. Second, *Betty Blocks* focuses less on the implementation of practices with third-party vendors. This makes it hard to implement practices such as *Prioritize Traffic*.

Pega (supporting 42 practices) is focused on letting their consumers build richer applications with their platform. These richer applications require integration with other applications. However, the focus of *Pega* is on in-house company projects that integrate within the company, or are complemented by selected organizations and partners. Companies are not supported in building an open platform that attracts complementors: capabilities for advanced community engagement or monetization strategies are not supported.

Overall the focus of the LCDPs appears to be on building enterprise applications, and less on platforms or even ecosystems. All LCDPs show that they have developed mature platforms, with support for modern standards in security and resource management. Through their implemented practices they enable their customers to develop and publish modern APIs that can be consumed by complementors. However, looking at the areas *Community* and *Commercial*, which contain less technical practices, we observe a gap. Many of the more advanced capabilities that customers can use to build platforms and attract complementors, such as *Monitoring*, *Analytics*, *Community Engagement*, and *Monetization Strategy*, are left to their customers to implement. The LCDPs support around 50% of the practices, leaving the other 50% to be implemented by their customers. In the evaluation of the API-m-FAMM with non-LCDP ecosystems we encountered five products that implemented respectively 42%, 59%, 42%, 77%, and 79% of these practices. Three out of five of these products are more mature than support offered by the LCDPs, meaning that customers would have to implement a number of practices themselves to built comparable products with one of the LCDPs.

Not all providers agree with our belief that they should support API management activities to enable their customers to create platforms. By not implementing these practices, and leaving them to be custom implemented, their customers have to invest more effort in building a platform on top of their LCDP. The providers miss the opportunity to support better API management for citizen developers. Instead they obligate citizen developers to seek help from professional developers to complete these tasks. This creates a dependency from citizen developers on these professional developers, and misses the opportunity to put more power in the hands of the citizen developers, democratizing software development even further. As claimed in the Gartner report [19] LCDPs improve the productivity and reduce the time to market, and because of the shortage of developers, democratizing development would offer a possible solution. However, the current state of these LCDPs does not enable citizen developers to create platforms, without requiring the support of professional developers. Raising the abstraction of API management practices could and should be the next step for the LCDPs.

6 Engineering Research Challenges for LCDPs

The previous section discussed the current state of API management support among LCDPs, measured with the API-m-FAMM. Even though we provide LCDP providers with engineering and product planning direction through this evaluation with the API-m-FAMM, there are still several research challenges that hamper further progress in this domain. These challenges are based on our observations made during the case studies and on the authors' experience with software ecosystems and LCDPs. They are based on the capabilities that show the highest number of practices that require a *custom* implementation, and common remarks extracted from the interviews. We outline these engineering research challenges here and provide several solution directions.

Life Cycle Management - Citizen developers will be constructing new application extensions and releasing them to customers, probably without regard for software and data complementors who use a previous version of the application. Citizen developers need to be made more aware of the effects of data model and interface changes for the software ecosystem surrounding the application. While typically these problems would be solved through abstraction, it is practically impossible for citizen developers to remain ignorant of the effects of software evolution on interfaces with third parties in the ecosystem. This can be accomplished through, often complementary, practices such as versioning policies, backwards compatibility, publishing road maps, and change notifications [4,6,12].

The LCDPs support impact analysis within the platform, knowing the relations between different components. However, novel solutions to support analysis of impact on applications outside of the LCDP are necessary to support the citizen developer.

Performance - Considering that the created applications will be approached through different channels than the traditional user interface, novel architectures are required that can handle large volumes of traffic through other channels, such as APIs. Architecture styles such as Microservices [7] offer a possible solution to these scalability challenges. Of course, an important requirement is the abstraction that citizen developers should be offered.

Observability - LCDPs should be able to handle an increase in users, while still providing the citizen developer with control over who uses the API, how much the API is used, and how the API is used. API gateways [4] traditionally provide these controls to professional developers and operational staff, but now need to be supported by the LCDP itself and usable by non-technical users. The citizen developers need to have access to API usage metrics and statistics to ensure that they too can identify misuse and monitor traffic from the citizen developer's partners [20].

Community - As the community around a product starts growing, complementors need to be supported as much as possible. Such capabilities are for example enabling citizen developers to generate API access credentials for complementors, infrastructures for communicating with complementors, as well as providing application stores around a generated product. All studied LCDPs leave the development of these community practices to the citizen developers.

The abstractions provided by LCDPs should give citizen developers enough control over API documentation and usage, while automatically adding technical documentation such as SDKs and source code examples. The studied LCDPs offered this through the use of standardized specification languages, such as the OpenAPI specification.

In the past, research has been conducted on the generation of APIs [15]. However, a number of related practices, such as *Provide FAQ and Code Samples* and *Provide Start-up Documentation*, are only offered through consumer built solutions. These practices are challenging to support due to an ever evolving generated object model. Without support from the LCDP the developer is responsible for evolving the manual written documentation together with the model of the API. This will lead to mistakes that hurt the community.

While the term 'citizen developer' indicates that it has been the goal to open up software engineering to people without formal software engineering education, we can hardly claim that this has been successfully accomplished for API management. The complexity of modern software solutions and the inherent simplification required to create LCDPs are constantly in direct conflict with each other. The platformisation trend lays this bare and shows that new models and perspectives are required to truly make software engineering accessible to any citizen developer. We see it as future work to design new abstractions that make LCDP solutions simpler and more powerful in supporting API management practices.

7 Threats to Validity

In this paper we present four descriptive case studies that we conducted based on interviews and documentation. Through these case studies we evaluate the current state of API management support offered by LCDPs.

Our conclusions are threatened by concerns regarding the generalizability of these four LCPDs when compared to the LCDP industry as a whole with respect to API management maturity. We cannot deny that there could be an LCDP that we did not study that supports more, or even all, API management practices. However, given that we studied four major platforms that are recognized as such in the Quadrant for Enterprise LCDPs report [19] confirms that we have studied a representable group. While the API maturity evaluation might not be generalizable to other LCDPs, these four platforms are recognized as the most innovative in the industry. Given that there might be a provider that has a more mature support of API management practices only confirms that these leaders and visionaries are missing out on opportunities to further support their customers. Providers with less mature support of API management practices make our call to action only more pressing.

Another threat to validity of this research are the evaluations of the LCDPs based on the API-m-FAMM. Wrong or imprecise evaluations based on documentation and interpretation could distort the conclusions. The fact that the interviewees reviewed and corrected the evaluation mitigates this risk. We believe that the general evaluation of the LCDPs with respect to API management, combined with the vision of the LCDP provider gives a truthful representation of the current state of API management maturity. Our findings and conclusions are based on the global state of API management support of the LCDPs, and do not depend on specific practice support.

In our research we focused on the LCDPs and their API management capabilities. Although we discussed a number of organizations that built an internal platform on top of the LCDP, we did not discuss specific example platforms. We did not specifically search for an example, but rather focused on the general state of API management maturity. Future work should study existing platforms built on LCDPs to further understand what opportunities LCDPs have.

8 Conclusion

Our case studies, as presented in Section 4, evaluate four LCDPs using the maturity model API-m-FAMM. Our research was guided by the research question: *How mature are the API management capabilities that LCDPs offer?* We conclude that these LCDPs support around 50% of the practices described in the API-m-FAMM. The other practices are left to be implemented by the customers of the LCDPs. We conclude that only *Mendix* places API management firmly on its road map. Both *Betty Blocks* and *Pega* do not observe a demand for API management capabilities among their customers, and neither are they promoting these capabilities. *OutSystems* recognized the demand, but has not yet focused on providing more of these capabilities to their customers. Instead they defer much of the work to either third-party vendors or the LCDP customers. By not supporting these practices we believe that LCDP providers miss out on the opportunity to further democratize software development. They instead require citizen developers to solicitate the support of professional developers to develop platforms that are open for other companies to extend.

We draw the following conclusions from this work. First, we suspect that LCDP providers will soon be challenged in providing capabilities that enable citizen developers to transform their applications into platforms. Our research shows that LCDP providers are currently unable to support such capabilities for citizen developers and require technical staff to implement such architectures and mechanisms through either third-party solutions or custom solutions built on top of the LCDP. Second, we conclude that as LCDPs are becoming more powerful, they can use the API-m-FAMM to evaluate and update their road maps. Finally, we identify five engineering challenges that, if solved, will create a next generation of citizen developers who can independently create complete software platforms and software ecosystems, and subsequently manage them without the requirement for highly specialized technical knowledge.

References

1. Andreo, S., Bosch, J.: API management challenges in ecosystems. In: International Conference on Software Business. pp. 86–93 (2019). https://doi.org/10.1007/978-3-030-33742-1_8
2. de Bruin, T., Rosemann, M., Freeze, R., Kulkarni, U.: Understanding the main phases of developing a maturity assessment model. ACIS 2005 Proceedings - 16th Australasian Conference on Information Systems (2005)
3. Cabot, J.: Positioning of the low-code movement within the field of model-driven engineering. Proceedings - 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS-C 2020 - Companion Proceedings pp. 535–537 (2020). https://doi.org/10.1145/3417990.3420210
4. De, B.: API Management. Apress, Berkeley, CA (2017). https://doi.org/10.1007/978-1-4842-1305-6_2
5. de Feijter, R., Overbeek, S., van Vliet, R., Jagroep, E., Brinkkemper, S.: DevOps competences and maturity for software producing organizations. In: Enterprise, Business-Process and Information Systems Modeling. vol. 318, pp. 244–259. Springer Verlag (2018). https://doi.org/10.1007/978-3-319-91704-7_16
6. Hora, A., Robbes, R., Valente, M.T., Anquetil, N., Etien, A., Ducasse, S.: How do developers react to API evolution? A large-scale empirical study. Software Quality Journal **26**(1), 161–191 (2018). https://doi.org/10.1007/s11219-016-9344-4
7. Jamshidi, P., Pahl, C., Mendonca, N.C., Lewis, J., Tilkov, S.: Microservices: The journey so far and challenges ahead. IEEE Software **35**(3), 24–35 (2018). https://doi.org/10.1109/MS.2018.2141039
8. Jansen, S.: A Focus Area Maturity Model for Software Ecosystem Governance. Information and Software Technology **118**(November 2019), 106219 (2020). https://doi.org/10.1016/j.infsof.2019.106219
9. Jansen, S., Brinkkemper, S., Cusumano, M.A.: Software ecosystems: Analyzing and managing business networks in the software industry. Edward Elgar Publishing (2013). https://doi.org/10.4337/9781781955635
10. Mathijssen, M., Overeem, M., Jansen, S.: Identification of Practices and Capabilities in API Management: A Systematic Literature Review. Tech. rep., Utrecht University (2020), `http://arxiv.org/abs/2006.10481`

11. Mathijssen, M., Overeem, M., Jansen, S.: Source Data for the Focus Area Maturity Model for API Management. Tech. rep., Utrecht University (2020), `https://arxiv.org/abs/2007.10611v3`
12. Medjaoui, M., Wilde, E., Mitra, R., Amundsen, M.: Continuous API Management: Making the Right Decisions in an Evolving Landscape. O'Reilly Media (2018)
13. Overeem, M., Jansen, S., Fortuin, S.: Generative versus interpretive model-driven development: Moving past 'It depends'. In: Pires, L., Hammoudi, S., Selic, B. (eds.) Model-Driven Engineering and Software Development. MODELSWARD 2017. Comm. in Comp. and Inf. Science, vol. 880, pp. 222–246. Springer International Publishing, Cham (2018). https://doi.org/10.1007/978-3-319-94764-8_10
14. Poell, T., Nieborg, D., van Dijck, J.: Platformisation. Internet Policy Review **8**(4), 1–13 (2019). https://doi.org/10.14763/2019.4.1425
15. Polák, M., Holubová, I.: REST API management and evolution using MDA. In: C3S2E '15: Proceedings of the Eighth International C* Conference on Computer Science & Software Engineering. pp. 102–109 (2015). https://doi.org/10.1145/2790798.2790820
16. Ralph, P., et al.: Empirical Standards for Software Engineering Research (2021), `https://arxiv.org/abs/2010.03525`
17. Sanchis, R., García-Perales, O., Fraile, F., Poler, R.: Low-code as enabler of digital transformation in manufacturing industry. Applied Sciences (Switzerland) **10**(1) (2020). https://doi.org/10.3390/app10010012
18. Van Steenbergen, M., Bos, R., Brinkkemper, S., Van De Weerd, I., Bekkers, W.: The design of focus area maturity models. In: International Conference on Design Science Research in Information Systems. vol. 662, pp. 317–332. Springer, Berlin (2010)
19. Vincent, P., Iijima, K., Driver, M., Wong, J., Natis, Y.: Magic Quadrant for Enterprise Low-Code Application Platforms. Tech. Rep. September, Gartner (2019)
20. Weir, L.: Enterprise API Management: Design and deliver valuable business APIs. Packt Publishing Ltd (2019)