A Use Case-based Investigation of Low-Code Development Platforms

Robin Lichtenthäler, Sebastian Böhm,
Johannes Manner, and Stefan Winzinger

Distributed Systems Group, University of Bamberg, Germany
{robin.lichtenthaeler,sebastian.boehm,
johannes.manner,stefan.winzinger}@uni-bamberg.de

Abstract. Rapid application development without profound development skills are the stated advantages of the recent trend in Low-Code Application Development. In a time-constrained experiment we investigate these promises for three Low-Code platforms by implementing a practical use case. While this was in fact feasible in a short time for major parts of our use case, the platforms differ significantly and a technical understanding is still required for non-trivial applications.

Keywords: Low-Code, Process Automation, Use Case, Mob Programming

1 Introduction

Low-Code application development is a recent trend in the software industry. It is expected to become the technological basis for an increasing amount of newly developed applications, as predicted in the *2021 Gartner Magic Quadrant for Enterprise Low-Code Application Platforms* [9]. The promised advantages of Low-Code platforms are that on the one hand less skills are required for development and on the other hand applications can be developed much faster [6], because Low-Code platforms enable application development at a higher level of abstraction, often based on visual programming [6]. By integrating cloud computing, Low-Code platforms can furthermore support application deployment in an automated fashion on reliable and scalable cloud infrastructures [8]. A variety of platforms have emerged from both vendors specialized on Low-Code (e.g., OutSystems[1], Mendix[2], or Appian[3]) and established cloud providers (e.g., Microsoft[4]). Nowadays, the interest in Low-Code platforms also increases in academic research. Several studies focusing on single platforms in-depth have been published [2, 5, 7] and we are aware of one study [8] aiming to compare several Low-Code platforms to provide a broader overview of the existing platforms and

[1] https://www.outsystems.com
[2] https://www.mendix.com
[3] https://appian.com/
[4] https://powerapps.microsoft.com

J. Manner, D. Lübke, S. Haarmann, S. Kolb, N. Herzberg, O. Kopp (Eds.): 14th ZEUS Workshop, ZEUS 2022, Bamberg, held virtually due to Covid-19 pandemic, Germany, 24–25 February 2022, published at http://ceur-ws.org

their scope of features. Also evaluations of how well certain features facilitate application development have been performed. Henriques et al. [2] evaluated the process modeling language of OutSystems in a structured way and Sahay et al. [8] discussed their experiences as a side aspect. An investigation focusing specifically on the aspect of rapidness of development while considering several platforms in comparison, has however not been done yet. Therefore, we performed a qualitative investigation of several platforms by implementing a realistic use case in a constrained experimental setup. We aim to investigate the promises of easy, rapid application development and deployment by evaluating to which extent this is possible with a selected set of platforms. This is summarized by our research question:

RQ: To what extent do Low-Code Development platforms enable rapid application development despite low prior experience?

In the following, we describe foundations of Low-Code platforms and our use case in Sect. 2. Our approach is explained in Sect. 3. In Sect. 4 we state the outcomes of our investigation and discuss them with regard to our research question in Sect. 5, before concluding our work in Sect. 6.

2 Foundations

Low-Code Development Platforms are cloud environments in which applications can be created and hosted based on the technology stack provided by the platform vendor. The core characteristic is that vendors try to abstract as much as possible from the technical details (the coding aspect) of application development [3, 8] while still enabling developers, called *citizen developers* in this context [3, 8], to implement their specific use case. While a higher level of abstraction and a more constrained development environment simplify application development, the degree to which applications can be customized for a specific use case is limited. However, a more complex development environment is more difficult to use but would allow for a specific customization. Therefore, platform vendors apply a range of techniques to balance this trade-off, namely visual programming [4, 8], domain-specific languages [3], model-driven engineering [1, 3], or pre-built components and templates [3]. Internally, platform vendors make use of cloud computing to allocate resources for applications on-demand and in an automated and scalable way as described in more detail by Sahay et al. [8]. They also provide a comparison of current Low-Code Development Platforms [8]. Differences between the platforms originate from the diverse backgrounds of platform vendors. Regarding suitable use cases and applications, Luo et al. [6] found out that most developers, who discuss such platforms online, build mobile and web applications. A focus is on process automation [6] and business-centric applications [6], such as Customer Relationship Management (CRM) or Content Management Systems (CMS).

From our point of view, our use case therefore fits well into the context of Low-Code applications, as we want to automate the process of a student registering for a thesis at a university's chair. Similar to CRM, the student can be seen as a kind of customer managed by the chair and similar to CMS different documents are

involved to describe the thesis topic and for the registration of a thesis. We have modeled the use case in its current manual form using Business Process Model and Notation (BPMN) to identify potentials for automation[5]. Our envisioned process includes a web-based application which manages and automates parts of the process. The beginning of this process is shown in Fig. 1. For a better understanding, the process can be divided into three Sub Processes (SP): Request Submission (SP1), Chair Notification (SP2) and Acceptance Decision (SP3).

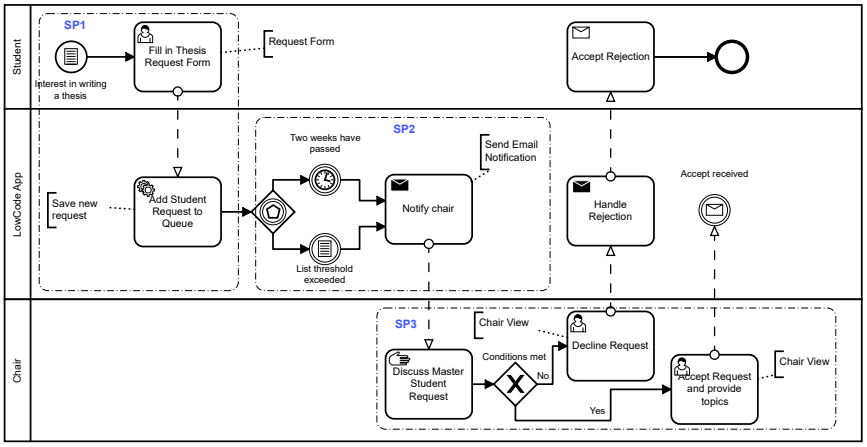

Fig. 1. Thesis Registration Process at the University of Bamberg (excerpt).

Although this excerpt is only the beginning of the whole thesis registration process, it is representative for the functional requirements of the whole process, covering all layers of a typical web application: User interfaces (UI), data storage (DS) and business logic (BL). Starting in SP1, at the UI level, students should be able to fill in a **Form** with their personal information, such as name, email, and course of studying. In addition, they should be able to upload a file that contains further information or ideas for potential thesis topics. The app should validate the submitted form and **Save** the **Data** in a storage layer, enriched with the date of submission and with a status "New". In SP2 at the BL layer, there are two possible events which represent **Triggers** for a notification. If either two weeks have passed since the submission of the oldest request or the list of new requests has exceeded a certain threshold, the app should detect this and thereupon send an **Email** to notify the members of the chair that a meeting should be held to discuss the new requests. Finally, in SP3, after a chair meeting, it should be possible for a chair member to **View** a list of all requests and to either decline or accept each request through the click of a button. The **Logic** layer behind it

[5] `https://github.com/uniba-dsg/low-code-use-case`

should validate that only new requests can be accepted or declined and change the status accordingly by performing an **Update** on the **Data** in the DS layer.

3 Methodology

Our methodology mainly follows an experimental approach as we perform a qualitative study by implementing the previously introduced process model. However, for a general understanding and market overview of Low-Code platforms, we searched for empirical comparisons by using ACM, IEEE, and Scopus.

Our literature search yielded a comprehensive comparison of Low-Code platforms by Sahay et al. [8]. From their list of platforms, we selected those that are still available, offer *Process Automation* as a feature, and can be tested within a free tier offering. Hence, we selected OutSystems, Microsoft Power Apps, and Appian because only these fulfill our criteria. For the implementation, we strictly followed the process model. We documented the implementation process for all platforms, w.r.t. the perceived achievements and obstacles, structured by the defined subprocesses. We applied *Remote Mob Programming*[6] as a development technique. The authors of the study were also its participants. This technique offers distributed collaboration for a small team of developers (at least three). Hereby, the members of the team are working together at the same time remotely, e.g., via video conferencing. One by one shares the screen with the current development environment and works actively as a so-called Typist who is executing instructions given by the rest of the Mob. In regular intervals (we set 15 minutes instead of the recommended 10 minutes), the advance in the development is handed over to the next member who becomes the Typist. We followed this procedure for three days by spending 6h/day (in line with the recommendation) for each of the platforms. Remote Mob Programming as a development technique was selected because we believe that compared to an approach of working with a new platform alone, the Mob can prevent situations of a single developer getting stuck. In such situations others can contribute their ideas and discuss the next steps or potential solutions to overcome this. This can accelerate the development process and discharge the involved developers. For the assessment, we used a 3-step scaling system to rate the degree of fulfillment for our requirements. All Mob members had no prior experience with Low-Code platforms in general. Only preparations like account creation and software installation were performed beforehand.

4 Implementation Results

In this section, we summarize the results of the experiment which can be seen in Table 1. This includes the aspects of our use case implemented and the problems that occurred during implementation.

[6] `https://www.remotemobprogramming.org/`

Table 1. Subprocesses from Figure 1 implemented: Fully implemented (●), partially implemented (◐), and not implemented (○).

	SP1 (Request)		**SP2 (Notification)**		**SP3 (Decision)**		
	Form	Save Data	Triggers	Email	View	Logic	Update Data
Appian	◐	◐	◐	●	◐	●	●
OutSystems	●	●	◐	●	●	●	●
PowerApps	●	●	◐	◐	◐	○	○

4.1 Appian

A unique characteristic of Appian is its custom scripting language. All building blocks (views, data, logic) are objects in the appian designer. The integration between these objects was hard to grasp at the beginning of our experiment. Some features were only realizable when implementing custom boolean expressions or configuring elements in a way where the graphical editor had no predefined option and we had to use the scripting language. This complicated the development process in our one-day-workshop. Reading the documentation and understanding the scripting language was often necessary to proceed. Appian integrates with public cloud providers and their systems as well as to other external sources like ERM/CRM systems. A deployment was implicitly done when storing the different elements.

The UI implementation was well supported via the graphical editor. Some aspects were not implementable with the editor, e.g., when more mature features like grouping input fields (SP1.Form) or a display of all students with their status buttons (SP3.View) were needed. The logic is defined in a process-like interface similar to the options in BPMN where tasks can be configured via the scripting language. Triggers can start these processes. For mail integration, we used the predefined Appian service. The timer trigger did not work in our experiment whereas the element size check of the already stored entities succeeded (SP2.Trigger). We stored primitives through our forms, but faced difficulties storing files (SP1.Save Data). With our gained knowledge, SP3 was implemented without issues.

4.2 OutSystems

In contrast to Appian and PowerApps, OutSystems requires locally installed client software which include self-explanatory hints, tooltips and autocompletion features. All study participants used the same account with the same log-in credentials. Only the desktop application was used since the synchronization worked properly via the cloud. Particularly noteworthy was the deployment which could be done by a single click without any further knowledge.

After some time, we found out, that forms can be created based on the data model. OutSystems' storage solution uses one of several predefined SQL servers where files can be stored as blob entries in a table. Therefore, we defined the data model first and created our views by attaching the data table to it. Additional

fields of other tables were created via dragging and dropping the corresponding input fields from the tool bar. The input fields were automatically integrated in a submit form which already included the logic to persist the data. OutSystems was the only platform, where we managed the mentioned file upload and download (SP1.Save Data). The time based trigger did not work in our use case, whereas the *list threshold exceeded* trigger did (SP2.Trigger). The mail integration in SP2 was possible with an existing SMTP server. A consistent look and feel of the platform helped to implement SP3 without facing any challenges.

4.3 PowerApps

PowerApps is fully integrated with other Microsoft products and the Azure cloud. To make use of these integrations, code writing skills are required for different layers of the Low-Code platform (e.g., for validation or UI workflows). The documentation, however, is structured on an individual product level, making it difficult to find details and best practices on the integration with PowerApps. Another difficulty were unclear error messages, e.g., "Schemas do not match" or "HTTP request failed" in the context of integrating a Flow with a UI button.

For SP1, the form and data storage was realizable after some starting problems. Data storage (Dataverse) and the business logic execution (Flows engine) are separate, modular products which complicate building a full stack app in our experimental setting. As for the other platforms, business logic can be executed based on triggers which were difficult to integrate with the Dataverse tables. Triggering the mail worked partially, see Table 1, as we were able to send a mail based on a timer but not on the exact conditions of the list threshold. Mail integration was possible with an existing SMTP server. Due to the problems we faced, especially with the integration of the different products, there was not enough time left to implement SP3, apart from creating the *accept* and *decline* button in the view.

5 Discussion

As an initial answer to our research question we can state that we were able to rapidly realize large parts of our use case with all three platforms, despite no previous specific experience with them. Nevertheless, our general knowledge about software engineering helped us in finding the needed options in the platforms more quickly. Being familiar with the classic three-layer application design, for example, helped us to already have a general structure for the application in mind. From our point of view, someone without general software engineering knowledge had to follow more closely the guidelines, tutorials, and documentation provided by the platform to become productive but could, after some training, also profit from the features of a platform to build applications rapidly. However, as we could see from the more detailed aspects of our use case (e.g., data validation or checking custom conditions), a higher use case complexity usually requires more technical knowledge which is also in line with the overall assessment of

Luo et al. [6]. Such technical knowledge might stem from a general background in computer science or from practical experience with a platform over a longer time. Detailed knowledge for one platform, however, might be difficult to apply elsewhere. That's why, from our point of view, a general background in computer science still provides a better foundation for application development. In addition to these general findings, the three platforms we investigated can be further differentiated. Overall, we found OutSystems to be a solution more focused on *citizen* developers. Its clean design via drag and drop elements, automatic integration with the storage layer and in particular its one-click-deployment contribute to the best Low-Code experience we faced in this experiment. The other two solutions could be described as *Less*-Code platforms. PowerApps is integrated with a lot of services of the Microsoft Azure's cloud environment. For enterprise users, this integration might be beneficial since already used services can be integrated into PowerApps, but programming is more technically compared to the application stack in OutSystems. Contrary to the other platforms, Appian offers a custom scripting language together with a custom component model. For feature-rich, rapid bootstrapped applications this platform design might support customers to build web applications. Without deeper technical knowledge in Appian and an understanding of the DSL, Appian is an expert system as the market for professional Appian developers shows.

Regarding features and techniques that supported rapidness in application development, we faced the best experience whenever the low code platforms provided intuitive interactions, suggestions, and direct feedback. One example are software wizards for generating UIs based on the defined data scheme which guide through the creation process in a focused interface. Additionally, these interfaces rely on visual programming by offering drag and drop features to alter the UI for displaying some fields respectively leaving out others, all while arranging these fields in a visually appealing layout as a default. In these cases, no extensive software engineering skills are needed to understand the layering of a system and progress can be made quickly, because the platform provides suitable defaults and templates. In other cases where custom DSL expressions need to be written the implementation flexibility increases, but comes with the difficulty to be familiar with the specific syntax. To take low coding literally, we refined the classification scheme for our selected platforms. We only rate OutSystems as a *low* code platform whereas Appian and PowerApps are *less* code ones.

Considering the feasibility of implementing our use case with the three platforms, we can state that from a functionality point of view, all platforms fulfill the functional requirements. In comparison to other implementation options, additional aspects like vendor lock-in or overall operation costs would need to be considered, but are out of the scope of this work. Finally, our experiment also has some threats to validity: (1) All study participants aka paper authors have a background in computer science. Therefore, the comparison to citizen developers is flawed and could be investigated in an additional experiment in future work. (2) The time limitation of six hours for each platform is one strength for a fair comparison but can introduce false conclusions due to the shallow investigation

of the documentation and an absence of deeper knowledge about the platforms and their designs. (3) Some features like encryption of the data, security and role management with providers like LDAP were not tested due to time limitations.

6 Conclusion

In this paper, we made an experimental investigation of Low-Code platforms. We find that the investigated Low-Code platforms do enable rapid application development through easing the implementation effort. Nevertheless, coding is still required at some points, which is shown by the two Less-Code platforms in our investigation. Low-Code platforms are within this area of tension between ease of usage, enabling citizen developers to create apps, and the integration with other systems and a cloud provider's ecosystem. Due to this trade-off, some platforms are focused on a single application for development whereas others provide a feature-rich ecosystem.

References

1. France, R., Rumpe, B.: Model-driven development of complex software: A research roadmap. In: Proc. of FOSE (2007), https://doi.org/10.1109/FOSE.2007.14
2. Henriques, H., Lourenço, H., Amaral, V., Goulão, M.: Improving the developer experience with a low-code process modelling language. In: Proc. of MODELS (2018), https://doi.org/10.1145/3239372.3239387
3. Khorram, F., Mottu, J.M., Sunyé, G.: Challenges & opportunities in low-code testing. In: Proc. of MODELS Companion (2020), https://doi.org/10.1145/3417990.3420204
4. Kuhail, M.A., Farooq, S., Hammad, R., Bahja, M.: Characterizing visual programming approaches for end-user developers: A systematic review. IEEE Access 9, 14181–14202 (2021), https://doi.org/10.1109/ACCESS.2021.3051043
5. Lebens, M., Finnegan, R.: Using a low code development environment to teach the agile methodology. In: Agile Processes in Software Engineering and Extreme Programming, pp. 191–199. Springer International Publishing (2021), https://doi.org/10.1007/978-3-030-78098-2_12
6. Luo, Y., Liang, P., Wang, C., Shahin, M., Zhan, J.: Characteristics and challenges of low-code development. In: Proc. of ESEM (2021), https://doi.org/10.1145/3475716.3475782
7. Martins, R., Caldeira, F., Sa, F., Abbasi, M., Martins, P.: An overview on how to develop a low-code application using OutSystems. In: Proc. of ICSTCEE. IEEE (2020), https://doi.org/10.1109/ICSTCEE49637.2020.9277404
8. Sahay, A., Indamutsa, A., Ruscio, D.D., Pierantonio, A.: Supporting the understanding and comparison of low-code development platforms. In: Proc. of SEAA (2020), https://doi.org/10.1109/SEAA51224.2020.00036
9. Wong, J., Iijima, K., Leow, A., Jain, A., Vincent, P.: Magic quadrant for enterprise low-code application platforms. online (2021), https://www.gartner.com/doc/reprints?id=1-27IIPKYV&ct=210923&st=sb, published 20 September 2021

All links were last followed on January 18, 2022.