Low-code platforms and languages: the future of software development

Gabriel Juhás*‡‖, Ludovít Molnár†‡, Ana Juhásová¶, Miriam Ondrišová§, Milan Mladoniczky*‖, Tomáš Kováčik*‖
*Faculty of Electrical Engineering and Information Technology
Slovak University of Technology, Bratislava, Slovakia
gabriel.juhas@stuba.sk
†Faculty of Informatics and Information Technologies
Slovak University of Technology, Bratislava, Slovakia
ludovit.molnar@stuba.sk
‡Interes.Institute s.r.o., Bratislava, Slovakia
interes@interes.institute
§Faculty of Arts
Comenius University in Bratislava, Slovakia
miriam.ondrisova@uniba.sk
¶ BIREGAL s.r.o., Bratislava, Slovakia
ana.juhasova@biregal.sk
‖ NETGRIF, s.r.o., Bratislava, Slovakia

2022 20th International Conference on Emerging eLearning Technologies and Applications (ICETA) | 979-8-3503-2033-6/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICETA57911.2022.9974697

***Abstract*—Low-code as a term in software development was mentioned by Forrester in 2014 [1] to denote development platforms, that make application development more accessible to broader community of developers by simplifying the application development process. To simplify development, low code platforms apply several different principles, that have even much older roots, such as visual programming, model driven development with different types of diagrams, process and workflow modelling and management. As a result low-code platforms offer some kind of application builders or creators that allows to drag and drop diagrams and forms to develop the final application, which is then running within a platform engine, often as a service in cloud. The aim of the low code development platforms is on one hand to solve the problem with shortage of human resources in IT by making application development more accessible and on other hand to make application development faster. The hope of both low-code development platforms producers and enterprises as their consumers is to make digital transformation and automation much more accessible and effective. According to Gartner [2], the share of new applications developed by organizations using low-code platforms should reach 70% of all new applications developed by organizations in 2025, compared to less than 25% in 2020. If the Gartner prediction will come true, then low-code will be the future of software development. One of the biggest drawbacks of low code development platforms is that they either hide the (parts of) code from developers of generate the code in lower level languages, that are not accessible for low-code developers. To remove the drawback, one need to define a low-code programming language that will be used to store the source code of the final application generated by the application builder of the platform. Then the platform application engine should serve as an interpreter (as a virtual machine) of the low-code programming language. Although most of today low-code platforms do not work with a low-code language, there are some pioneers. For example, in 2021 Microsoft released an open-source low-code language, called PowerFx [3], for its low-code development platform Power Apps, originally released during 2016, Zoho low-code platform uses their own proprietary language called Deluge [4], Netgrif low-code platform uses open-source low-code language Petriflow [5]. In this paper, we discuss the paradigms that low-code programming languages should implement, such as abstraction from tiers in multi-tier architecture. We also discuss our experience how to teach low-code principles.**

I. Introduction

In analogy to human languages, that serve people to tell each other what to do, programming languages serve people to tell computers what to do. In other words, programming languages enable people to command computers. In analogy to writing that is used to encode human languages, source code is used to encode programming languages, resulting in so called source code.

Based on its hardware, computers can handle machine code while the human language is natural choice for people. Therefore the history of programming languages is basically formed by attempts to cover the gap between low-level machine code and human language by definition of higher level languages, starting by assembler, and continuing by higher level languages such as C, C++ or Java, including compilers and/or interpreters from higher-level to lower-level languages, as illustrated in Figure 1. There are also attempts to cover the gap from opposite direction, using different modeling techniques and languages, such as UML, BPMN, that are easier to understand by humans.

The basic principle of the higher level programming languages is that they abstract from some implementation details, that are present in the lower-level language to which the higher level programming language is compiled or interpreted. As a result, with each and every higher level one gets a simpler code, easier to understand for humans, when compared with the the lower-level code. Starting from 2014, when Forrester

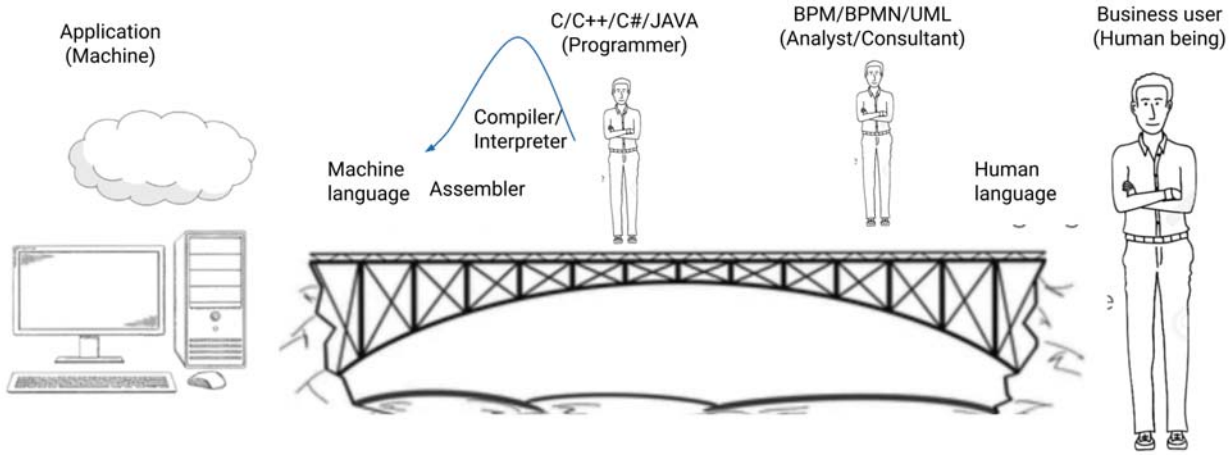

Fig. 1. A human computer communication model

[1] introduced the term low-code to denote the software development platforms that simplify application development by, for example visual programming principles, different kinds of diagrams and/or drag-and-drop data form designers, the low-code development platforms become still more popular, predicted to have 75% share of new applications developed in 2025 [2].

Low-code platforms successor denoted as no-code platforms, promises application development with no code writing. The motivation for low-code and no-code development is given not only by effort to simplify and faster application development, but at the same time to make it more accessible for broader community of application developers. When this effort will eventually be successful, it will solve the problem with lack of human resources. According to [6], in 2021 there were 26.8 millions of active software developers among world population of almost 7.9 billions [7], which is less than 3.5%. On the other hand, the global shortage of the software engineers in predicted to exceed 85 millions by 2030 [8].

Despite the decades of the effort in defining higher level programming languages that will make programming more accessible, one of the reasons of this shortage of manpower is still great complexity of present programming languages used in today application development. One could make an analogy with the literacy in ancient Egypt: one of the reasons of low literacy in ancient Egypt (some sources assume 1-5% [9], but the exact number is not important) is due to the complexity of hieroglyphs. But history has shown, that the complexity of writings in ancient Egypt did not lead to no writings. Instead, the writing was simplified in mankind history leading to modern alphabet used to write in modern languages. This, together with other factors such as improved access to education and improved methods of education lead to high level of literacy in modern world. Although this analogy is far from perfect, one could conclude that the complexity of coding applications in today programming languages will not lead to no coding, but rather to simplified coding abstracting from implementation details. Thus, instead of no code one should focus on creation of low-code programming languages that will enable to write simpler code. Then, application builders of the next generation low-code platform (or even no-code platforms) should generate the code of the application in such low-code languages and application engines of the next generation low-code platforms should interpret the resulting low-code applications.

There are already some pioneers of such low-code languages and next generation low-code platforms supporting those low-code languages, such as open-source low-code language PowerFx [3] supported by low-code development platform Power Apps from Microsoft, language Deluge [4] supported by Zoho low-code platform or open-source low-code language Petriflow [5] supported by Netgrif low-code platform. In next section we discuss the paradigms that low-code programming languages should implement, such as abstraction from tiers in multi-tier architecture, concept of a user and concept of a task and support of queries.

979-8-3503-2033-6/22/$31.00 ©2022 IEEE

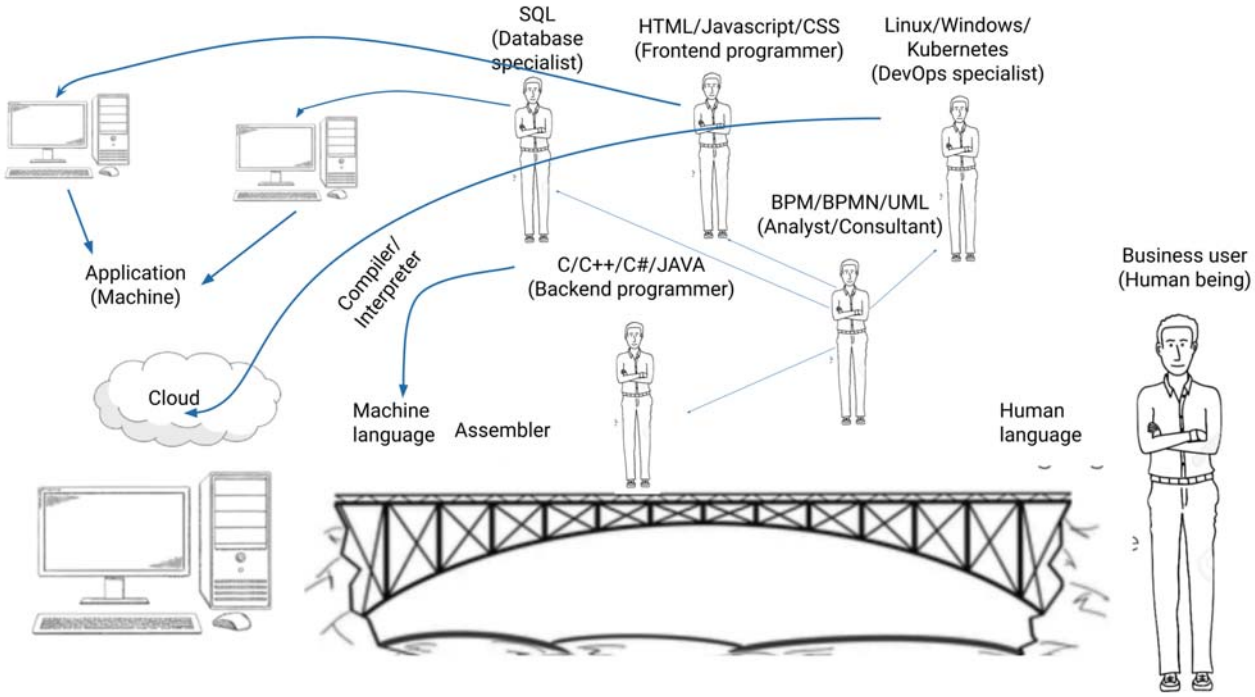

Fig. 2. A refined human computer communication model

II. Principles and Requirements for Low-Code Languages and Low-Code Platforms

A. Abstracting from layers in multi-layer architecture

The requirements for low-code languages and low code application development platforms heavily depend on the application domain. Requirements for virtual reality applications will differ from requirements for business to customer (B2C) application or business to business (B2B) applications that digitalize business processes and services. When breaking down today B2C and B2B applications that are result of custom development, such as eBanging, eFinance, eInsurance, eCommerce applications, Customer care systems, or applications supporting process automation in general, these are typically multi-layer or multi-tier application, consisting usually of data layer, application layer (back-end, server) and presentation layer (front-end, client).

Many of such applications use an SQL database for data layer, java for application layer and JavaScript and HTML5 for presentation layer. Thus, for example, when a business requirement tells that in such an application one need a data variable for a surname of a person, then at presentation layer one need to define HTML input with ID surname, then to define JavaScript variable surname, to write a piece of code that will transfer surname value using HTTPS protocol using e.g. POST request to the application layer written in java.

To speed up application development, instead of writing application layer in pure JavaScript and HTML5 with styling in CSS, one uses a framework to code presentation layer, such as Angular. To code application layer, one typically uses a framework too, for example Spring Boot. To process surname value from application layer, one writes a controller that process the POST, create a Java Bean for person and save the value of the surname into an attribute surname in created Java Bean object for person. Controller processing the POST request uses a method of an implementation of a service interface for a person to save person bean with the surname value. This method call a save method of an extension of JPA Repository (or CRUD Repository), that will persistently save the person object with the surname value into database using annotations that maps the Java Bean person to a table person of relational database creating a row in the table and saving the value of the surname in the column surname of the table. As JPA is only a specification, in fact one uses some implementation of JPA such as Hibernate framework or uses JDBC instead of JPA. The chosen languages, specifications and their implementations and used frameworks that cover all layers and their communications represent a full stack.

To manage development process, one typically uses software and services such as GitHub or BitBucket that can effectively manage the collaboration of a team of developers, including managing pull requests to merge changes and managing branches and versions of the application.

Once the application is developed, one need to deploy it on an infrastructure, which can basically be either a physical or virtual server with appropriate operating system, e.g. a

979-8-3503-2033-6/22/$31.00 ©2022 IEEE

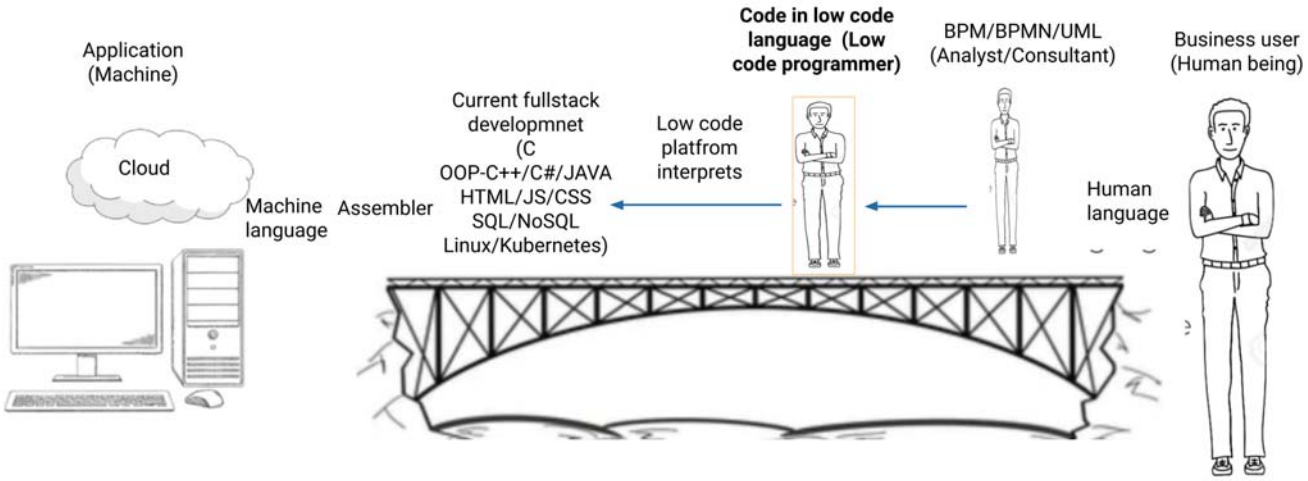

Fig. 3. A human computer communication model with next generation low-code platforms interpreting low-code languages

distribution of Linux, and with web and application server, e.g. Apache Tomcat, and database server, e.g. MySQL Server. The application also can be containerized for example in a Docker, and the containers can be scaled in orchestration software, typically Kubernetes.

We try to visualize the languages and developers of multi-layer application development in the Figure 2.

To successfully develop such complex applications with numerous languages and frameworks used, one either need full stack developers of specialist covering each layer. The complexity of the multi-layer application resulting in the need of literacy in broad spectrum of programming languages and frameworks in one of the reason for the shortage of the human resources in application development.

Thus, the first and maybe the most urgent requirement on low-code language is to abstract from multi-layer architecture when defining data variables and their manipulation, in analogous way in which higher level languages such as Java abstracted from operational memory (register) allocation when defining data variables, compared to machine code. The resulting situation for low-code languages abstracting from layers in multi-layer architecture is depicted in Figure 3.

Let us mention here that to simplify deploying of applications written in a low-code language we encourage that low-code platform engine is an interpreter rather than a compiler, mainly because once deploying a low-code platform engine itself using standard (costly) deployment, no further deployment is necessary when uploading low-code application to low-code platform interpreting the low-code. From the perspective of deployment, the most effective option would be low-code platform as a service, because it would reduce the deployment costs for users of the service.

B. Combination of data and processes: data objects with life-cycle

When looking to the history of Computer Science and Information Technologies Industry one can observe that the primary programming languages were concentrated on algorithms and flowchart. First later the orientation on data arises, resulting in relational databases and object oriented programming languages.

Recently, when looking from business side of the bridge between computers and humans in Figure 3, workflow models and business process models became more popular. Instead of choosing between data orientation and process orientation, a low-code language should support both concepts equally. Relational databases and Object oriented programming (OOP) languages support just very primitive explicit life-cycle of data objects.

In databases these are represented by well known CRUD operations forming a simple life-cycle starting with Create, followed by possibility to execute Read and Update arbitrary many times and then optionally Delete a record.

In OOP there is a constructor, then methods of the objects can be called and optionally there is an explicit destructor or a garbage collector. However, nor SQL nor OOP languages does

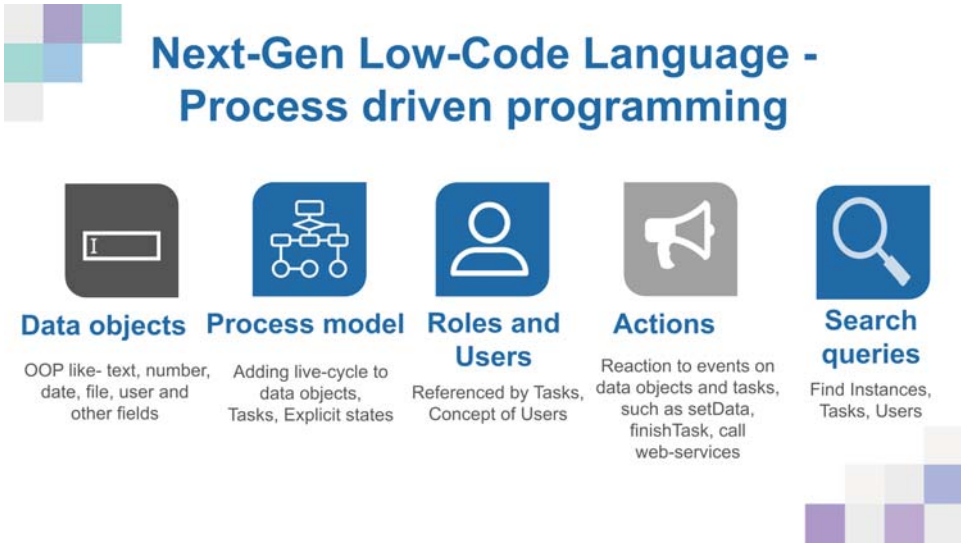

Fig. 4. Requirements on low-code languages

not support explicit life-cycle in form of a workflow process for data objects.

Here we advocate to combine OOP with Business Process Modeling by adding a life-cycle in form of a workflow process to data in low-code languages. Any kind of workflow process models that support explicit states and build processes as flows of tasks as basic blocks can be used. Thus, a class of low-code objects should consists of data variables, life-cycle of objects in form of a workflow process consisting of tasks as basic building blocks as well as data references telling for each task which data variables can be read or changed in given task.

C. Event handling, actions, users and roles

Most of the web-based multi-layer applications are reactive event driven systems, where users click on buttons, select a menu item, fill the input field, upload a file etc. Thus, to enable process automation using a low-code language, one should include events and users into the low-code language. If one have data variables and tasks as basic building elements of low-code objects, one should at least include an event for data reading and an event for changing data. For a class of low-code objects itself, one should include at least an event for upload of the class to a low-code platform, an event for creation of an object of the class and an event for deletion of an object of existing object of the class. It also should provide an API for user management, including creation of Roles as sets of users with possibility to add and remove a user from a Role. For task, the should be at least following events: an event for assigning a task to a user, an event that cancel assigned task and an event that finish assigned task. Once a task is assigned to users, the users can trigger events for reading and changing data referenced to the assigned task. It should be possible to low-code what to do, i.e. how to react, when an event is triggered by the user. In particular, it should be possible to react by triggering another event. For example, as a reaction to an event changing a data field loan amount triggered by a user, the event that change (recompute) the monthly payment of the loan should be triggered.

D. Search queries and basic views

One of the most important part of the low-code language should be a simple yet powerful query language, that will enable to find low-code objects according to selected search criteria, to find tasks fulfilling selected search criteria and to find users fulfilling selected search criteria.

The client of the engine of a next generation low-code platform that interpret applications in a low-code language should enable to organize search queries as filters via menu items. It should also provide a view displaying collection of selected objects, and a view displaying collection of selected tasks when choosing corresponding menu item. In addition, the front-end of the engine of the low-code platform should provide a view for displaying executable tasks of a low-code object.

The requirements on low-code languages described in previous subsections are summarized on Figure 4.

III. Next Generation Low-Code Platform Architecture

One of the most important feature of today low-code platform, that are not supported by any low-code language, is that they enable to create an application using visual programming, dragging and dropping data forms etc. This features should be preserved in next generation low-code platforms that will generate and deploy applications in a low-code language. It should also be supported to reuse existing models by importing data models in form of entity-relationship diagrams or SQL scripts as well as importing process models in form of BPMN. A next generation low-code platform should provide an application builder that will generate code of the application

979-8-3503-2033-6/22/$31.00 ©2022 IEEE

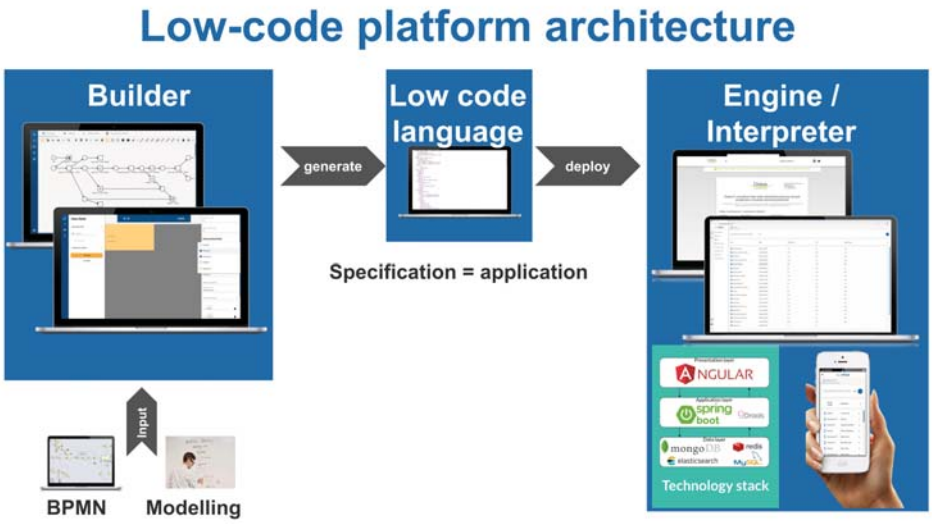

Fig. 5. Next generation low-code platform architecture

in the supported low-code language. The generated application in low-code language should be interpreted in an engine of the next generation low-code platform, as depicted in Figure 5.

IV. Low code platforms over low code languages as solution to business needs

When comparing different ways of digital transformation and process automation, businesses are interested in time to market and customization capabilities.

Businesses usually already have some outdated information systems from the past, often referred as legacy systems. The necessary changes in such legacy systems are often time consuming and customization capabilities are limited. To automate business processes, one can either buy so called off-the-shelf applications, that are ready to use and that automate a concrete domain, such as invoice processing of customer relationship systems. Time to market for process automation using an off-the-shelf application is fast, but the customization are still limited. This leads to the situation that the process to automate must be often accommodated to the off-the-shelf application.

On the other hand, with full-stack custom applications business reach high degree of customization, but due to high complexity of the full-stack the time to market is much slower compared to off-the-shelf applications.

Today low-code development platforms aims to achieve faster time to market compared to full-stack custom application, and higher level of customization capabilities compared to off-the-shelf applications. But the customization capabilities of today low-code/no code applications are still significantly lower compared to the full-stack custom applications, and off-the-shelf applications time to market is still significantly shorter than time to market of today low code/no code applications.

The next generation low code platforms, that relay on a sound low-code language lower the time to market and increase the customization capabilities, as depicted in Figure 6.

A. Source code availability, customization, extensions and plugins

Due to abstraction in low-code languages, there level of customization capabilities of next generation low code platforms will still remain a lower than in the case of full-stack custom applications. In order to increase the customization capabilities of next generation low-code platforms, the availability of the source code of the platform engine and possibility to override its functionality would help.

However, overriding some functionality may slow time to market for the future, because of the need to override the functionality in updates and new versions of the platform engine again and again. To avoid this and still enable new custom functionality, concept of plugins that can be written in any language and whose functionality can be called in low-code language should be added. Such plugins should be kept separately from the engine of the platform itself, and it should be possible to add a plugin without necessity to re-deploy the engine. Such plugins can add a custom functionality to back-end of the engine.

To enable customization of the front-end, a concept of pre-defined components and templates, that can be copied, customized and registered by the engine. Similarly to the possibility to call functionality of back-end plugins in low-code, it should be possible to specify in low-code that for displaying a component on a client device the customized registered component should be used.

One can go even further and enable extensions of low-code language itself without necessity to extend the platform engine itself, just using plugin management end events. Such extension can be realized by defining specification of the extension and by writing a translation from the extended

979-8-3503-2033-6/22/$31.00 ©2022 IEEE

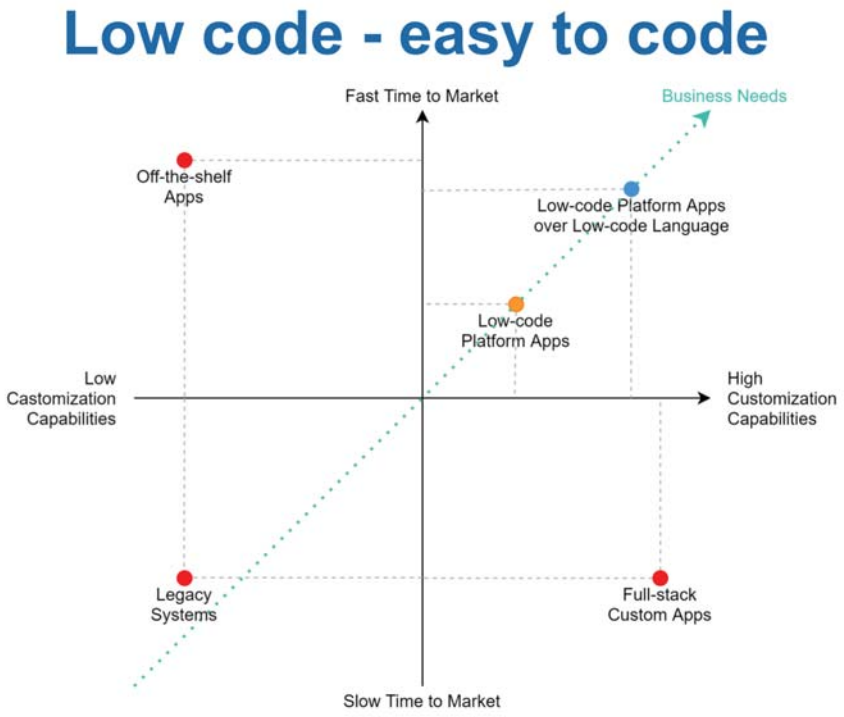

Fig. 6. Trade-off between customization capabilities and time to market

low-code language to the original low-code language. Such translation can be registered as a plugin and called in an action by event for uploading the extended low-code object.

Thus, when coding an application, we follow the principle "as low-code as possible and as full stack custom code as necessary" to achieve desired level of customization, while strictly following separation of concerns principle between full stack code and low-code.

V. Experiences with low-code applications and education

We have defined petriflow [5] as a low-code language that fulfills the previously mentioned requirements and implemented Netgrif low-code platforms [10] consisting of Netgrif application builder that generates Petriflow code and Netgrif Application Engine that interprets Petriflow code.

In fact, the definition of the Petriflow and the implementation of the Netgrif platform was highly influenced by repeating client requirements, that lead us to the generalization and recommendations on low-code platforms over low-code languages as presented in this paper. Netgrif low-code platforms interprets dozens of low-code applications in different domains, including automation of business processes in Insurance, Leasing, Utility, Healthcare and Public Services. Our experience confirms the customization capabilities and time to market as depicted in Figure 6, mainly in case of frequent change requests on running applications.

In parallel, we teach courses that uses Petriflow language and community edition of Netgrif low-code platform at Faculty of Electrical Engineering and Information Technology of the Slovak University of Technology in Bratislava. We also teach various courses covering full stack application development, including courses on OOP using Java as a programming language. When comparing the learning curve and the ability to create application in full stack code and low-code it is clearly in favor of low-code. Interestingly, also the quality of the resulting application developed by the same student in favor of low-code. Our hypothesis is that it is caused that because the platform engine itself is written as a full stack application following the best practices in full stack application development and the students concentrate entirely on application logic written in low-code language, while when developing the full stack application the students often do not strictly follow the best practices of full stack application development, and because of the complexity of the full stack, concentrate less on application logic itself.

Using low-code language that attaches life-cycle of objects as a workflow process also enable to concentrate on process patterns, such as approval processes, or evaluation processes.

To our knowledge, Petriflow low-code language and Netgif low-code platform are used in courses at several other study programs, e.g. at FernUniversität in Hagen, Germany and Faculty of Informatics and Information Technologies of the Slovak University of Technology in Bratislava.

979-8-3503-2033-6/22/$31.00 ©2022 IEEE

VI. Summary

In this paper we discussed requirements on low-code languages as a next generation languages that abstract from full stack coding and therefore enable to develop application significantly faster when compared with full stack custom application development and are easier to master for students. The presented set of requirements and principles of low-code languages is definitely not complete. The requirements and principles of low code languages and low code platforms supporting such languages are derived from our experience with the desing, the usage and with the teaching of Petriflow low-code language and Netgrif low-code platfrom that supports Petriflow language.

References

[1] C. Richardson and J. Rymer, "New development platforms emerge for customer-facing applications," 2014. [Online]. Available: https://www.forrester.com/report/New-Development-Platforms-Emerge-For-CustomerFacing-Applications/RES113411

[2] Gartner, "Gartner says cloud will be the centerpiece of new digital experiences," 2021. [Online]. Available: https://www.gartner.com/en/newsroom/press-releases/2021-11-10-gartner-says-cloud-will-be-the-centerpiece-of-new-digital-experiences

[3] Microsoft, "Microsoft power fx overview," 2022. [Online]. Available: https://learn.microsoft.com/en-us/power-platform/power-fx/overview

[4] Zoho, "The power of next generation programming," 2022. [Online]. Available: https://www.zoho.com/deluge/

[5] G. Juhás, T. Kovácik, J. Kovár, M. Kranec, and L. Petrovic, "Petriflow language and netgrif application builder," in *Proceedings of the Demonstration Resources Track co-located with the 19th International Conference on Business Process Management, BPM 2021*, vol. 2973. CEUR Workshop Proceedings, 2021, pp. 171–175.

[6] SlashData, "Developer nation report," 2022. [Online]. Available: https://www.developernation.net/developer-reports/dn21

[7] U. Nations, "A world of 8 billion: Towards a resilient future for all - harnessing opportunities and ensuring rights and choices for all," 2021. [Online]. Available: https://www.un.org/en/observances/world-population-day

[8] T. Phillip, "Is there a shortage of developers? developer shortage statistics in 2022," 2022. [Online]. Available: https://codesubmit.io/blog/shortage-of-developers/

[9] U. C. London, "Literacy," 2003. [Online]. Available: https://www.ucl.ac.uk/museums-static/digitalegypt/literature/literacy.html

[10] G. Juhás, T. Kovácik, J. Kovár, M. Kranec, and L. Petrovic, "Netgrif application engine." in *Proceedings of the Demonstration Resources Track co-located with the 19th International Conference on Business Process Management, BPM 2021*, vol. 2973. CEUR Workshop Proceedings, 2021, pp. 166–170.