# Journal of Computer Languages

## A novel model for Low-Code platforms development

--Manuscript Draft--

| | |
|---|---|
| **Manuscript Number:** | COLA-D-23-00045 |
| **Full Title:** | A novel model for Low-Code platforms development |
| **Article Type:** | Full Length Article |
| **Keywords:** | Low-Code; Microservices Architecture; Microservice-Driven Development; Microservice-Specific Language; Model-Driven Development |
| **Corresponding Author:** | Mehdi Ait Said, Ph.D.<br>Hassan 1st University Faculty of Science and Technology Settat<br>MOROCCO |
| **Corresponding Author Secondary Information:** | |
| **Corresponding Author's Institution:** | Hassan 1st University Faculty of Science and Technology Settat |
| **Corresponding Author's Secondary Institution:** | |
| **First Author:** | Mehdi Ait Said, Ph.D. |
| **First Author Secondary Information:** | |
| **Order of Authors:** | Mehdi Ait Said, Ph.D. |
| | Abdellah Ezzati, Phd |
| | Sara Arezki, Phd |
| **Order of Authors Secondary Information:** | |
| **Abstract:** | This paper presents a novel Low-Code Platforms (LCP) development model, which is a result of the adoption of a novel MSA approach in two software companies over three years. To validate the efficacy of our model, it was implemented in a third software company and evaluated by ten experts. The evaluation results confirmed that our model can overcome five LCPs' challenges: customization limitations, integration complexity, vendor lock-in, security concerns, and the difficulty of maintaining LCPs. Furthermore, the technical debt and learning curve associated with the implementation can be mitigated if the environment is already familiar with MDD and MSA. |
| **Suggested Reviewers:** | Salah E. Loukili, Phd<br>Software Engineer, Oracle Cerner<br>salah.loukili@gmail.com |
| | Soukaina MIHI, Phd<br>Software Engineer, Hassan 1st University Faculty of Science and Technology Settat<br>soukaina.mihi@uhp.ac.ma |
| | Mohamed Hassoun, Phd<br>Software Engineer, Capgemini Engineering<br>mohamed.hassoun@capgemini.com |
| **Opposed Reviewers:** | |
| **Additional Information:** | |
| **Question** | **Response** |
| **Free Preprint Service**<br><br>Do you want to share your research early as a preprint? Preprints allow for open access to and citations of your research | NO, I don't want to share my research early and openly as a preprint. |

| | |
|---|---|
| prior to publication.<br><br>Journal of Computer Languages offers a free service to post your paper on SSRN, an open access research repository, when your paper enters peer review. Once on SSRN, your paper will benefit from early registration with a DOI and early dissemination that facilitates collaboration and early citations. It will be available free to read regardless of the publication decision made by the journal. This will have no effect on the editorial process or outcome with the journal. Please consult the SSRN Terms of Use and FAQs. | |
| To complete your submission you must select a statement which best reflects the availability of your research data/code. IMPORTANT: this statement will be published alongside your article. If you have selected "Other", the explanation text will be published verbatim in your article (online and in the PDF).<br><br>(If you have not shared data/code and wish to do so, you can still return to Attach Files. Sharing or referencing research data and code helps other researchers to evaluate your findings, and increases trust in your article. Find a list of supported data repositories in Author Resources, including the free-to-use multidisciplinary open Mendeley Data Repository.) | Other (please explain: e.g. 'I have shared the link to my data/code at the Attach File step'). |
| You have selected other. Please explain: as follow-up to "To complete your submission you must select a statement which best reflects the availability of your research data/code. IMPORTANT: this statement will be published alongside your article. If you have selected "Other", the explanation text will be published verbatim in your article (online and in the PDF).<br><br>(If you have not shared data/code and wish to do so, you can still return to Attach Files. Sharing or referencing research data and code helps other researchers to evaluate your findings, and increases trust | I have shared the link to my data |

| | |
|---|---|
| in your article. Find a list of supported data repositories in Author Resources, including the free-to-use multidisciplinary open Mendeley Data Repository.)" | |
| To complete your submission you must select a statement which best reflects the availability of your research data/code. IMPORTANT: this statement will be published alongside your article. If you have selected "Other", the explanation text will be published verbatim in your article (online and in the PDF).<br><br>(If you have not shared data/code and wish to do so, you can still return to Attach Files. Sharing or referencing research data and code helps other researchers to evaluate your findings, and increases trust in your article. Find a list of supported data repositories in Author Resources, including the free-to-use multidisciplinary open Mendeley Data Repository.) | Other (please explain: e.g. 'I have shared the link to my data/code at the Attach File step'). |
| You have selected other. Please explain: as follow-up to "To complete your submission you must select a statement which best reflects the availability of your research data/code. IMPORTANT: this statement will be published alongside your article. If you have selected "Other", the explanation text will be published verbatim in your article (online and in the PDF).<br><br>(If you have not shared data/code and wish to do so, you can still return to Attach Files. Sharing or referencing research data and code helps other researchers to evaluate your findings, and increases trust in your article. Find a list of supported data repositories in Author Resources, including the free-to-use multidisciplinary open Mendeley Data Repository.)" | I have shared the link to my data |

# A novel model for Low-Code platforms development

Mehdi AIT SAID[1], Abdellah EZZATI[2], and Sara AREZKI[3]
{m.aitsaid[1], abdellah.ezzati[2], sara.arezki[3]} @uhp.ac.ma

**1: (The corresponding author)** Faculty of Sciences and Technologies, Hassan First University, Morocco, Settat, Km 3, B.P. : 577 Route de Casablanca
**2:** Faculty of Sciences and Technologies, Hassan First University, Morocco, Settat, Km 3, B.P. : 577 Route de Casablanca
**3:** Faculty of Sciences and Technologies, Hassan First University, Morocco, Settat, Km 3, B.P. : 577 Route de Casablanca

**Abstract.** In today's fast-paced and highly connected world, software development has become an increasingly critical aspect of modern society, from banking and finance to healthcare and education. Software plays an essential role in shaping the way we live, work, and interact with each other. As a result, the competition between companies is now on more than just the cost and quality level but also on the development speed, companies have to deliver more software in a shorter time. This led to the appearance of Low-Code platforms (LCP), a new style of software development, but it is still limited in some business domains due to several challenges. Our objective is to surpass these challenges by proposing new models and approaches for LCPs development.

This paper presents a novel LCPs development model, which is a result of the adoption of a novel Microservices Architecture (MSA) development approach named Microservice-Driven Development in two software companies over three years. This model provides the software companies with a strong foundation to build their own LCPs, using a Microservice-Specific Language for a declarative integration of the MSA into LCPs. To validate the efficacy of our model, it was implemented in a third software company and evaluated by ten experts. The evaluation results confirmed that our model is capable of overcoming five LCPs' challenges: customization limitations, integration complexity, vendor lock-in, security concerns, and the difficulty of maintaining LCPs. Furthermore, the technical debt and learning curve associated with the implementation can be mitigated if the environment is already familiar with Model-Driven Development and MSA.

**Keywords**: Low-Code; Microservices Architecture; Microservice-Driven Development; Microservice-Specific Language; Model-Driven Development.

## 1. Introduction

In today's modern society, software consumption has become an integral part of our daily lives. From smartphones and laptops to smart home appliances and vehicles, we are surrounded by software that facilitates and enhances our interactions with the world around us. As a result, the software has transformed the way we live, work, and communicate with each other, and its influence is only growing stronger.
At the end of 2022, Statista[1] estimates that there will be 16.8 mobile devices [1] and about 21.5 billion connected devices worldwide [2]. As a result, software companies have become a driving force behind innovation, economic growth, and social change. From small startups to multinational corporations, software companies are at the forefront of developing cutting-edge technologies and solutions that shape the way we live, work, and interact with each other. Software companies face immense pressure to innovate and deliver cutting-edge solutions that meet the needs of modern society. Furthermore, with the rapid pace of technological change and the increasing demands of customers, companies must constantly adapt and evolve to stay relevant in the market.

We asked ten successful software company's CEO about today's success factors in the software market, and eight from ten agreed that the delivery time is the most critical factor for winning software project deals and the shorter the period in which the client receives the first demo, the better the chances of winning the project. Competition is usually based on providing the best quality at the lowest cost and in the shortest time. With all the development tools, methodologies, technologies, and frameworks that we have today, it has become easier for companies to provide high-quality software with a high degree of usability, reliability, functionality, flexibility, maintainability, scalability, and security. IBISWorld[2] reports that the Point of Sale Software Developers industry in the US has experienced an average annual growth rate of 10.7% over the five-year period from 2018 to 2023 [3], this number will likely grow as software becomes increasingly essential in various industries and sectors and new software companies emerge to meet the demand; this means that the cost of software development will decrease and become almost equal, especially with the software companies specializing in a specific domain, e.g., education or health. That makes the delivery time the key factor for winning software development deals, especially the period in which the client receives the first demo, in some development teams, this period is called Time-To-Demo. This led the software development process to evolve at two levels (i): software architecture and (ii): development style.

In recent years, Microservice Architecture (MSA) has led software architecture evolution. In 2014 James Lewis and Martin Fowler defined MSA as an approach to developing software applications that emphasize the use of small, independent services that work together to provide the overall functionalities of the application [4]. In an MSA, an application is broken down into a collection of independent services that are developed, deployed, and managed separately [7]. Each service focuses on performing a specific task or function and communicates with other services through

[1] https://www.statista.com/
[2] https://www.ibisworld.com/

well-defined interfaces. By doing so, MSA enables organizations to adopt a DevOps culture and release new features and updates faster [8,29].
MSA has gained popularity recently due to its ability to increase software development's agility, flexibility, reusability, and scalability. By breaking down an application into small, modular services, developers can change specific application parts without affecting the entire system, this allows for faster development and deployment times and makes scaling the application as needed easier [5,6]. MSA offers a number of benefits over traditional monolithic architectures, including increased flexibility, reusability, scalability, and resilience, as well as improved developer productivity and agility [9,10,11]. MSA stands out from other software architectures due to its focus on the independence principle, which applies to the services in terms of functionality, technology, and organization [30,23]. Thus, microservices are designed to fulfill a single business requirement or technical capability [23]. However, it also introduces some challenges, such as increased complexity and the need for robust service communication and management. As a result, it is important for stakeholders to carefully consider the trade-offs of microservices architecture and determine whether it is the right approach for their specific needs [12,13,14].

At the development style level, Low-Code platforms (LCP) have gained significant attention in recent years as powerful tools for accelerating software development and reducing the complexity of building software. In essence, LCPs are software development tools that allow developers to build applications with minimal hand-coding, relying instead on human-friendly tools such as visual interfaces, drag-and-drop components, Domain-Specific Language (DSL), and pre-built templates and modules [15, 16, 18, 19]. This approach can reduce the time and effort required to build and deploy software applications while also enabling organizations to respond rapidly to changing business needs. In fact, LCPs incorporate Domain-Driven Design (DDD) and Model-Driven Development (MDD) principles as part of their approach [17]. For example, some platforms allow developers to build models of their applications using visual diagrams, which can then be automatically translated into working code. Nevertheless, LCPs come with several challenges and limitations that stakeholders need to be aware of before adopting them, some of these challenges are inherited from MDD, and others appeared with the LCPs, which will be detailed in the next section.

The contribution of this paper is the presentation of a novel LCPs development model, which was developed as a result of the adoption of the Microservice-Driven Development (MSDD) approach in two software companies over the span of three years. This model uses a Microservice-Specific Language (MSSL) to provide software companies with a strong foundation to create LCPs and was implemented in a third software company in order to evaluate its efficacy. Ten experts evaluated the results and confirmed that the model is capable of overcoming five main challenges that are present in LCPs: customization limitations, integration complexity, vendor lock-in, security concerns, and the difficulty of maintenance. Additionally, the technical debt and learning

curve associated with the implementation can be minimized if the environment is already familiar with MDD and MSA.

This study targets two groups of audience : (i) researchers looking to broaden their understanding of software development facilitation and automation. (ii) practitioners and businesses wanting to upgrade their abilities and equipment for producing all kinds of software.

The remainder of this paper is organized as follows. Section 2 is a background to illustrate the challenges and limitations of the LCPs. Section 3 presents related work. Section 4 outlines the research hypothesis and the methodology employed in this work. Section 5 describes the LCPs development model in detail with a case study in Section 6. Section 7 contains the evaluation of our model with a discussion in Section 8. Finally, Section 9 concludes the paper and proposes potential directions for future work.

## 2. Background

LCPs have the potential to transform the software development process, making it more accessible to non-technical stakeholders and enabling developers to focus on higher-level tasks such as business logic and data modeling. As a result, it is important for organizations to carefully evaluate the benefits and drawbacks of LCPs, and determine whether they are a good fit for their specific needs [19, 20]. This may involve considering factors such as the complexity of the application, the level of customization required, and the availability of skilled resources. Ultimately, LCPs have the potential to revolutionize the way that software is built and delivered, but they must be used in a thoughtful and strategic manner in order to fully realize their potential [21].

While LCPs offer many benefits, there are also several challenges that organizations may encounter when using them. Table 1 shows the list and brief description of these challenges. One major challenge is the potential for customization limitations, as LCPs typically rely on pre-built templates and modules that may not meet the specific needs of a particular application. This can lead to a lack of flexibility and may require additional customization efforts outside of the LCP. Another challenge is the risk of vendor lock-in, as organizations may become reliant on a particular LCP and face difficulties migrating to another solution. Additionally, concerns around data security and compliance may arise, as LCPs often involve sharing sensitive data with third-party vendors, also there is the potential for reduced control and transparency in the software development process, as LCPs may obscure the underlying code and limit developers' ability to fully understand and troubleshoot issues. Finally, the maintaining challenge, LCP inherits this challenge from MDD because LCPs form heavily on modeling, it can be challenging to maintain the models and keep them up-to-date with changes in the underlying codebase, this challenge is the potential disconnect between the generated code and the original model, this means that there may be a mismatch between the code that is generated by the model and the original model itself,

which can cause errors or inconsistencies in the software. This can happen if the generated code is not properly synchronized with the model, or if there are errors in the model itself that are reflected in the generated code.

| No. | Challenge | Description |
|---|---|---|
| CH1 | Customization limitations | LCPs may not provide the same level of customization as traditional coding. This may limit the ability to create highly unique and specialized applications. |
| CH2 | Integration complexity | Integrating with existing systems can be challenging due to the need for compatibility with various technologies and platforms. |
| CH3 | Vendor lock-in | LCPs may tie organizations to a particular vendor or technology, making it difficult to switch to a different platform if needed. |
| CH4 | Security concerns | Rapid application development, and use of pre-built modules and templates can introduce potential security vulnerabilities that need to be addressed. |
| CH5 | Technical debt | Using LCPs can sometimes result in technical debt, where software applications are built quickly but may require more maintenance and updates in the long run. |
| CH6 | Skillset limitations / Learning Curve | LCPs may require a different set of skills than traditional coding and may not be suitable for highly complex or specialized applications. |
| CH7 | Difficult to maintain | The challenge of repairing a model generator arises when the platform is updated or upgraded, potentially leading to incompatibilities with the previously generated project. |

Table 1: Challenges of LCPs

## 3. Related work

Microsoft Power Apps[3] is an LCP that allows businesses to build custom applications for web and mobile platforms. The platform is fully integrated with Microsoft 365, enabling users to leverage data from various Microsoft applications to build custom applications. Additionally, the platform includes AI Builder, templates, and connectors that make it easy for users to create intelligent applications without any coding experience. Microsoft Power Apps also enables businesses to build custom mobile applications for iOS and Android platforms, which can be deployed to the App Store and Google Play Store. The platform is built on the Azure cloud platform and provides

[3] https://powerapps.microsoft.com/

advanced security and compliance features to ensure that the custom applications are secure and compliant with industry standards. These features make Microsoft Power Apps a powerful LCP ideal for businesses looking to build custom applications quickly and easily.
Microsoft Power Apps has several pros and cons. On the plus side, the platform provides a user-friendly interface that allows users to create custom applications with little to no coding experience. Microsoft Power Apps is also fully integrated with Microsoft 365, enabling businesses to leverage data from various Microsoft applications to build custom applications quickly and easily. The platform offers a range of templates, connectors, and AI models that make it easy to create intelligent applications. Businesses can also build custom mobile applications for iOS and Android platforms, and the platform provides advanced security and compliance features to ensure the applications are secure and compliant with industry standards. However, there are also potential limitations to the platform, including limited customization options beyond the templates and connectors provided, reliance on the Microsoft ecosystem, limited third-party integrations, and a learning curve for new users. Ultimately, businesses should carefully consider these pros and cons before deciding to use Microsoft Power Apps.

Salesforce Lightning[4] is an LCP that enables the creation of custom business applications using drag-and-drop components. It was introduced by Salesforce as a modern user interface for their CRM (Customer Relationship Management) platform, it includes pre-built templates, integration with Salesforce services, and the ability to extend your applications using JavaScript. It offers a visual development interface called Lightning App Builder that allows you to drag and drop components to create custom pages, applications, and dashboards. This platform also includes Lightning Components, which are reusable building blocks that can be used to build custom functionality within your applications. Lightning Components are built using the Aura framework, which is a JavaScript framework designed specifically for Salesforce Lightning.
Salesforce Lightning provides a mobile-first approach to application development, which means that applications created with the platform are optimized for use on mobile devices. It includes a responsive design system that automatically adjusts the layout and user interface based on the screen size and device type.
However, there is still a learning curve associated with the platform, and customization options may be limited for some use cases. The cost of licensing fees may also be a barrier to entry for some organizations, and the platform is dependent on the Salesforce ecosystem. Additionally, Salesforce Lightning applications are designed to be used online, which limits offline capabilities, making it unsuitable for organizations that need to use applications in environments with limited or no internet connectivity.

[4] https://www.salesforce.com/campaign/lightning/

OutSystems[5] is an LCP that enables developers to build custom web and mobile applications quickly and easily. It offers a visual development environment that allows developers to drag-and-drop pre-built components to create custom applications, it includes pre-built templates, themes, and workflows that help developers to build applications faster. It also includes a library of pre-built connectors that enable developers to integrate their applications with other systems and data sources.

The platform is built on the OutSystems Service Studio, which is a visual development environment that allows developers to create custom applications using a drag-and-drop interface. It includes a visual workflow builder, a visual data modeler, and a visual code editor that enable developers to build custom business logic without writing code.

OutSystems also includes a mobile application development platform that enables developers to create custom mobile applications for iOS and Android devices. The mobile platform includes a range of pre-built templates and components that make it easy to create custom mobile applications.

One of the key benefits of OutSystems is its ability to integrate with other systems and data sources. The platform includes a range of pre-built connectors that enable developers to integrate their applications with other systems and data sources, including Salesforce, SAP, and Microsoft Dynamics.

However, there are some limitations to the platform. The learning curve can be steep for some developers, and customization options may be limited for certain use cases. Additionally, the cost of licensing fees may be a barrier to entry for some organizations, and there is a certain amount of dependency on OutSystems for ongoing support and updates. Finally, while OutSystems offers mobile development capabilities, it may not be as robust as other dedicated mobile app development platforms, which may be a limitation for organizations that need to build highly customized mobile applications.

Mendix[6] is an LCP that enables developers to build and deploy custom business applications quickly and easily. With its drag-and-drop interface, Mendix allows users to create applications without needing to write any code. It has been recognized as a leader in the LCP market by several industry analysts, including Gartner and Forrester[34. 35]. It is used by companies of all sizes and industries.

Mendix offers a range of features and tools to support app development, including visual modeling, automated testing, and collaboration tools. It also offers a range of pre-built templates and connectors to popular enterprise software systems like SAP[7], Salesforce[8], and Microsoft Dynamics[9].

[5] https://www.outsystems.com/
[6] https://www.mendix.com/
[7] https://www.sap.com/
[8] https://www.salesforce.com/
[9] https://dynamics.microsoft.com/

The Mendix platform is designed to support agile development practices and includes features like version control, continuous integration and delivery, and deployment automation. It also supports a range of deployment options, including on-premise, cloud, and hybrid environments. It is particularly well-suited for developing complex enterprise applications that require integrations with multiple systems and data sources. Its visual modeling capabilities allow users to create complex data models, user interfaces, and business logic without needing to write any code.
However, Mendix can be expensive, and its drag-and-drop interface can be limiting for developers who prefer more control over their code. There is also a learning curve involved in mastering the platform, and there is a risk of vendor lock-in since Mendix is a proprietary platform.

Appian[10] is an LCP that enables developers to build and deploy custom business applications with ease. With its drag-and-drop interface, Appian allows users to create applications without needing to write any code, it offers a range of features and tools to support app development, including visual modeling, automated testing, and collaboration tools. It also offers a range of pre-built components and templates that make app development fast and efficient.
The Appian platform is designed to support agile development practices and includes features like version control, continuous integration and delivery, and deployment automation. It also supports a range of deployment options, including on-premise, cloud, and hybrid environments.
One of the key features of Appian is its ability to integrate with other enterprise systems and data sources. Appian offers a range of connectors to popular enterprise software systems like SAP, Salesforce, and Microsoft Dynamics, making it easy to build applications that integrate with these systems, it also includes a range of analytics and reporting tools, enabling users to analyze data and gain insights into their business operations.
However, the cost of Appian may be a barrier for some companies, particularly smaller ones with limited budgets. Appian's limited control and learning curve may also be drawbacks for some developers who prefer more control over their code. Furthermore, the limited customization options and the risk of vendor lock-in should be considered when using this platform.

Zoho Creator[11] is an LCP that allows users to build custom applications without needing to write any code. The platform includes a drag-and-drop interface and a range of pre-built components that make app development fast and efficient, it offers a range of features to support app development, including visual modeling, automated workflows, and data integration capabilities. It also offers a range of deployment options, including on-premise, cloud, and mobile.
One of the key features of Zoho Creator is its ability to integrate with other Zoho apps, as well as with third-party tools. This makes it easy to build applications that integrate with existing workflows and systems, it also includes a range of analytics and reporting tools, enabling users to gain insights into their business operations and make data-driven decisions. Additionally, the

[10] https://appian.com/
[11] https://www.zoho.com/creator/

platform includes a built-in scripting language that allows developers to add custom code to their applications if needed.
Zoho Creator is designed for businesses of all sizes and industries, and it offers a range of pricing options to suit different needs and budgets. It also includes a range of customer support options, including online documentation, and forums.
However, the platform's limited customization options and scalability may be a drawback for some users, particularly those with very large or complex applications. Additionally, we have noted that the platform's integration capabilities with third-party tools may be more limited than other LCPs.

Table 2 provides a brief comparison of existing work and our proposed approach, summarizing the core of each solution. As seen, all related works suffer from at least three challenges, except our proposed model which surpasses CH1, CH2, CH3, CH4, and CH7. However, CH5 and CH6 depend on the implementation environment of our model. If the environment is familiar with MSA and MDD, then CH5 and CH6 can be surpassed.

| Related work | CH1 | CH2 | CH3 | CH4 | CH5 | CH6 | CH7 |
|---|---|---|---|---|---|---|---|
| Microsoft Power Apps | No | No | No | Yes | Yes | No | Yes |
| Salesforce Lightning | No | No | No | Yes | No | No | Yes |
| OutSystems | No | Yes | No | Yes | Yes | No | Yes |
| Mendix | No | Yes | No | Yes | Yes | No | Yes |
| Appian | No | Yes | No | Yes | No | No | Yes |
| Zoho Creator | No | No | No | Yes | Yes | No | Yes |
| Proposed Model | Yes | Yes | Yes | Yes | Yes/No | Yes/No | Yes |

Table 2. A brief comparison of related work and our proposed work.

## 4. Hypothesis and Methodology

Our research design follow the Design Science (DS) [32] and the Design Science Research Methodology (DSRM) proposed by Peffers et al. [33]. DS is an approach to developing models, methods, and implementations that are intended to serve human purposes. DSRM adapts this to research in the area of information systems.

Our proposed model is a result of three years of MSDD adoption in two software companies. This section will provide an overview of the MSDD approach and explain how we gathered the necessary data to develop our model, as well as how companies can benefit from this data when implementing it.

The early stage of this work was based on the analysis of 107 MDD-based projects from two software companies to detect modeling patterns of MDD-based business software in order to create and train a Machine Learning model to integrate an Artificial Intelligence code generator directly into LCPs. Still, we noticed that MSA-based projects have a large percentage of repeated code, which means the integration of MSA with MDD can reach a high degree of reusability. Florian et al. address the possibility of this collaboration between MDD and MSA; they argue that MDD techniques such as abstraction, model transformation, code generation, modeling viewpoints, and languages can improve the characteristics of MSA [22].

Drawing inspiration from the same hypothesis proposed by Florian et al in 2018 [22], we proposed MSDD (Microservice-Driven Development) as a novel development method for software companies, in order to assess the impact of integrating the MSA on the MDD environment and, consequently, its effect on Low-Code development. If the impact is positive on the development process, then we may find a way to combine these three concepts for surpassing the challenges and limitations of LCPs.

MSDD is based on two concepts: (i) the idea that MSA should be used as the main software architecture, not just a solution, that is because software engineers often use the principles and practices of MSA in large, complex software; (ii) all microservices must be developed by the MDD approach, and development teams have to create models that represent the microservice's requirements, design, behavior, and other aspects, and then generate the code and other artifacts from those models.
Our hypothesis was that by using MSA and MDD-based Microservices (MDDMS) in all types and sizes of software, resources and time can be saved during the software building process by boosting MSA reusability, which is based on our initial notes about MSA reusability in the early stage of this work.

On June 1, 2019, two software companies with sizes ranging from 100 to 500 employees partially adopted our approach, applying it to 25% of their respective development teams. All of the engineers involved were already familiar with MSA and had five or more years of experience in software engineering and the MDD approach.

The MDD and DDD have already been adopted by the two companies. Consequently, the microservices were created using MDD and the DDD extraction method was used for project decomposition into microservices for both companies.
Extracting microservices from a project specification using DDD involves identifying the various bounded contexts within the application and determining the appropriate microservices for each bounded context. This requires a deep understanding of the domain model and the interactions between different parts of the system [8, 26 ]. The key to extracting microservices from a project specification using DDD is to focus on the business requirements and domain model and to identify the appropriate bounded contexts and microservices based on these requirements [24].

All projects went through a process of development that followed these steps:

1. First, the software domain should be identified and analyzed to identify the different bounded contexts within the system. Each bounded context represents a specific area of functionality and should be encapsulated in a separate microservice.
2. Next, each bounded context should be further analyzed to identify the individual aggregates and entities within each context. This analysis helps to define the boundaries of each microservice and the interfaces between them.
3. Once the microservices have been defined and their interfaces identified, the creation of a platform-independent model (PIM) is started directly, which captures the requirements and specifications of the software system, along with the key concepts and relationships that are relevant to the domain. This PIM is then refined into a platform-specific model (PSM) that incorporates details about the target platform and technology stack, and that can be used to generate code or other implementation artifacts. The development teams have the flexibility to use any modeling tools such as Eclipse Modeling Framework[12], Rational Software Architect[13], and Enterprise Architect[14] to create and edit models of software systems, as well as for generating code and other implementation artifacts from the models. Each team is allowed to select the appropriate tool for them. The Computation-Independent Model (CIM) is not mentioned because is included in the first step.
4. Finally, the microservices should be deployed and managed using a container orchestration system, such as Kubernetes or Docker Swarm, to ensure scalability, reliability, and high availability.

This study spanned from June 1, 2019, to June 1, 2022. The obtained data from this study is available for download at https://msdd.org/msdd-data.html. The data is stored in a publicly accessible location to ensure that readers can access it without requiring any special permissions or access, and any stakeholder/company can make use of it. The data is composed of four tables:

[12] https://www.eclipse.org/modeling/emf/
[13] https://www.ibm.com/products/rational-software-architect-designer
[14] https://sparxsystems.com/

Table A displays the list of the examined projects; Table B displays the list of the extracted microservices categories; Table C displays the list of projects and the microservices categories that they are composed of; Table D displays the frequency of microservices categories used. This study involved the examination and analysis of 61 projects from 13 different domains (Table A).

We conduct our analysis by extracting and listing all used microservices from all developed projects. We then categorize and name these microservices according to the business logic and data they handle. Table B lists the categories that were extracted. Each of the 61 projects is composed of a set of microservices belonging to these categories (Table C), this is because the development of each project begins by identifying the core domains and their associated subdomains within the application, these subdomains can then be mapped to specific microservices that encapsulate the business logic and data for that subdomain. An example of this is the "User Management Service" category, which includes all microservices that deal with user authentication (e.g. logins, registration, password reset, roles, permissions, etc.).

The results in Table D demonstrate the frequency at which a variant of each category is used in all projects, both within the same domain and across domains (distinct domains), as well as when a microservice is reused without any modification as a pre-built microservice. The "All projects" column displays the frequency of usage of a category of microservices across all projects, the "Cross-domain" column displays the frequency of usage of this category of microservices across all 13 domains, the "In-domain" column displays the frequency of usage of this category of microservices in the same domain of 13 domains, and the "As a Pre-built" column shows the frequency of use of variants of this category of microservices without any extra coding.
Out of the 49 microservices categories, 47 are reused at least once, 39 in distinct domains, 36 are reused at least one time in the same domain, and 32 are reused without any extra coding, which means 95.91% of microservices categories have been reused, 79.59% in distinct domains, 73.46% in the same domain (at least once), and 65.30% without any extra coding.
As is evident, MSDD enables the creation of autonomous and decoupled services that can be used in multiple contexts or domains, which is a key aspect of MSA. By considering MDD-based MSA as the main architecture for all projects, rather than just as a solution, we can maximize the MSA reusability potential. Developers can benefit from reusability by writing microservices that can be reused in various projects and even in different business domains. This not only reduces the amount of time and resources needed for development but also leads to more reliable, higher quality software, as the code has been tested and improved over time. By reusing code, developers can prevent duplication of work and can use existing code to create new features.

Using the data in Table C, we encapsulated all variations of each microservice category into a single microservice, for example, we built a User Management Microservice that encompasses all features and functionalities of this subdomain from all projects. We built all these microservices based on MDD. We obtained a collection of 49 MDDMS, we named this collection Microservices

Hub (MH). By utilizing this collection, each project from the 61 examined projects (Table A) can be assembled quickly. Additionally, projects from similar domains can be easily constructed with some customization, even if they are new. For example, a Hospital Management System can be built by [MS1, MS2, MS3, MS5, MS10, MS14, MS17, MS19, MS22, MS23, MS24, MS25, MS26, MS33, MS34, MS35, MS40, MS42, MS45, MS46, MS47]. Furthermore, cross-domain microservices (Table D) can be employed to construct an entirely new project from a different domain.

As a result, the probability of the existence of a microservice in MH increases as more new projects with new microservices are developed, because every new microservice must be added to the MH, and every new feature must be added to an existing microservice within the MH. Since the new projects can benefit from the existing microservices in the MH without or with some customization, that increases the possibility of reducing the time and resources needed to develop a new project,
These MDDMS are not just individual blocks of code or models created by one developer or a team to be reused in a particular context; they are fully independent services which can be used independently in various contexts. Because of this, microservices can be shared among teams and even between companies. Therefore, the adoption of MSDD can improve and boost the MSA reusability, the advantage of this high degree of reusability is that software development can be done faster and at a lower cost, which is a positive effect.

Over a period of three years, we have observed how adopting MSDD in two software companies has affected the development process. We have concluded that MSDD can improve the MSA reusability, reduce development time, and increase the quality of the code. If we can automate this process, either partially or completely, we can further increase these benefits, and this is the contribution of our research that will be presented in the next section.

## 5. A novel model for Low-Code platforms development

### 5.1 Problem and solution

After three years of observing how MSDD has impacted the development process in two software companies, we have come to the conclusion that it can enhance MSA reusability, reduce development time, and increase code quality and that by creating a hub of pre-built ready-to-use and easy-to-customize microservices. These microservices must be updated and upgraded. With each new project, new microservices can be added or existing microservices can be upgraded, and that's what we called it: The MH feeds.
In theory, the larger the size of the MH, the less probable it is that a new project will be composed of microservices that do not exist in MH.

The MH needs to be continuously updated and upgraded with new microservices and features. Algorithm 1 outlines the necessary steps to keep it up-to-date. As dictated by MSDD, any project should be broken down into individual microservices. For this purpose, we have two cases to consider: first, when all the extracted microservices are already available in the MH, we must only check for any new features and, if so, upgrade the relevant microservice in the MH. Second, if the new project consists of a set of no existing microservices in the MH and/or existing microservices with new features, we must develop the new microservices and upgrade the existing ones. After that, we can generate the source code from model transformations, customize the UI/UX, build it, test it, and finally deploy the artifact.

In MDD, the graphical user interface is typically generated automatically from the models and code that make up the application. However, there are situations where customization of the GUI may be necessary or desirable. This is why UI/UX customization is done after the source code is generated in the MDD approach.

Algorithm 1 MSDD-based Projects Development Process.

Input: Project Specification and the microservices collection (MH).

Output: A collection of microservices with the deployment environment config file.

Functions dictionary:

[

D(Project): Decompose a project to a set of microservices,
C(MS): Get the category name of a microservice,
G(Category): Get a pre-built microservice from MH by category name,
R(MS, PreBuiltMS): Check if a microservice requires additional features,
DevIt(MS): Develop and test a new microservice,
Generate(): Generate the code source,
Customize(): Customize UI/UX,
Build(): Build an executable artifact and deployment environment config file,
Test(): Run all testing stages,
Deploy(): Deploy the project,
U(MS, PreBuiltMS): Develop, test, and add the new features to PreBuiltMS,
Push(PreBuiltMS, IsNew): Push a PreBuiltMS to the MH if the IsNew var is true that means this PreBuiltMS is a new microservice, else is an upgraded microservice

]

```
1: ProjectMsCollection ← D(NewProject)
2: for MSi ∈ ProjectMsCollection do
3:     CategoryName ← C(MSi)
4:     if CategoryName ∈ MH then
5:         PreBuiltMS ← G(CategoryName)
```

```
6:        if R(MSi, PreBuiltMS) then
7:          PreBuiltMS ← U(MSi, PreBuiltMS)
8:           Push(PreBuiltMS, False)
9:        end if
10:    else
11:       PreBuiltMS ← DevIt(MSi)
12:       Push(PreBuiltMS, True)
13:    end if
14: Generate()
15: Customize()
16: Build()
17: Test()
18: Deploy()
```

Fig 1 illustrates the structure of a project based on Algorithm 1. As depicted, a project consists of a collection of microservices, and each microservice consists of a number of features. To create the source code for each feature, a series of models are employed, and the executable artifact can then be generated from the source code. The development process of a project based on this structure is divided into three steps:

1. The microservices extraction
2. The model's identification from features or system requirements and pulling microservices from the MH
3. The auto-generation of source code and executable artifact

Step 1 involves breaking down the project into microservices each responsible for a specific business capability or domain. To identify microservices, the project system's functionality is analyzed to determine the various business capabilities or domains that it supports. These capabilities are then grouped into distinct microservices, with each microservice having its own well-defined boundaries and responsibilities. this process can be semi to fully automated by using decomposition algorithms such as (i) the DDD approach to software development that emphasizes the importance of understanding the business domain in which the software operates. With DDD, the system's functionality is divided into distinct domains, and each domain is implemented as a separate microservice [8, [24]; (ii) the dataflow-driven approach presented by Shanshan et al, is a semi-automatic approach for decomposing a dataflow diagram into microservice candidates that encapsulate business logics [25].
Step 3, is fully automated by the MDD process. Flowing the rule of MSDD, development teams must create a series of high-level models that represent different aspects of the software system

they are building, these models are then used to generate the actual code for the software, as well as other artifacts such as documentation, test cases, and user interfaces.
Step 2 always requires developers to intervene, as we mentioned we face two cases: If the microservice has already existed in the MH, it must be cloned and personalized manually. If, however, it does not exist, it must be developed, and in some cases, a new microservice can be developed from other microservices models - for instance, Student Management Service can be developed based on the User Management Service models.

Even with the high degree of reusability achieved by MSDD, the developer's intervention is still needed. We created a novel language framework called MSSL (Microservice-Specific Language) to reduce this intervention. This MSSL is a Low-Code language, it is both human-friendly and declarative, meaning it can describe MH microservices, their features, relationships, and interfaces. Developers don't have to interact directly with the MH or model. After the project is broken down into microservices, they just need to use MSSL to describe each microservice and the MSSL compiler will take care of interacting with the MH to obtain and customize the required microservices. Additionally, MSSL has the capability to generate new features by using generic models and combining models from different microservices in the MH to create new microservices, and eventually generate the necessary source code for them. The generic models are a set of built-in (MSSL) models that can be used in any context, such as a form builder, objects serializer, etc, these built-in generic models can be used to develop new features in a microservice or even create entirely new microservice.

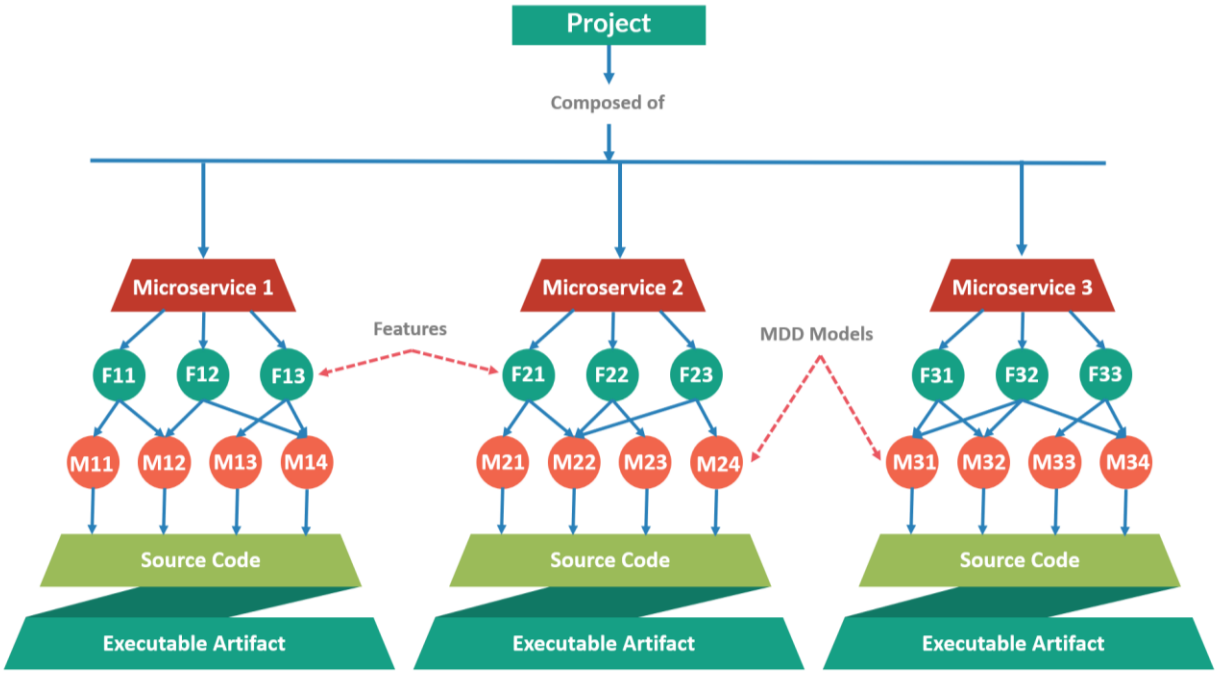


Fig. 1: The structure of MSDD-based projects

Fig. 2 illustrates the MSSL processing model. After checking and validating the syntax, the MSSL code will be processed by the MSSL compiler and communicated with the MH to export models from the target microservices and generic models. In fact, the MSSL compiler starts extracting the metadata of each microservice from the MSSL code. This metadata is stored in the compiler context, consisting of two blocs the Pulling Bloc (PB) and the Customizing Bloc (CB). The PB always contains the information of the target microservices in the MH such as name, version, etc. The CB includes two sub-blocs the Generic Metadata (GM) and the Targeted Metadata (TM). The GM is used to handle the generic model's customizations and the TM is used to handle the targeted microservices customizations.

The next step is using this PB to pull the desired microservices from the MH and customized them according to the GM and the TM by the MDD transformation concept to generate the Microservices Context (MC). Every microservice in the MC is a customized instance of a microservice in the MH, it's similar to pulling a Docker image from the Docker Hub and customizing it with a Dockerfile. Additionally, the MC serves as a live preview of the whole project in development mode, wherein all UI/UX personalization must be done. Once that is complete, the final source code can be exported and the executable artifact can be built.

The MC can be used as a configuration folder for projects. When we need to maintain or upgrade a project, we can simply import the MC folder into the MSSL Integrated Development Environment (IDE) and it will be loaded with all the customizations automatically.

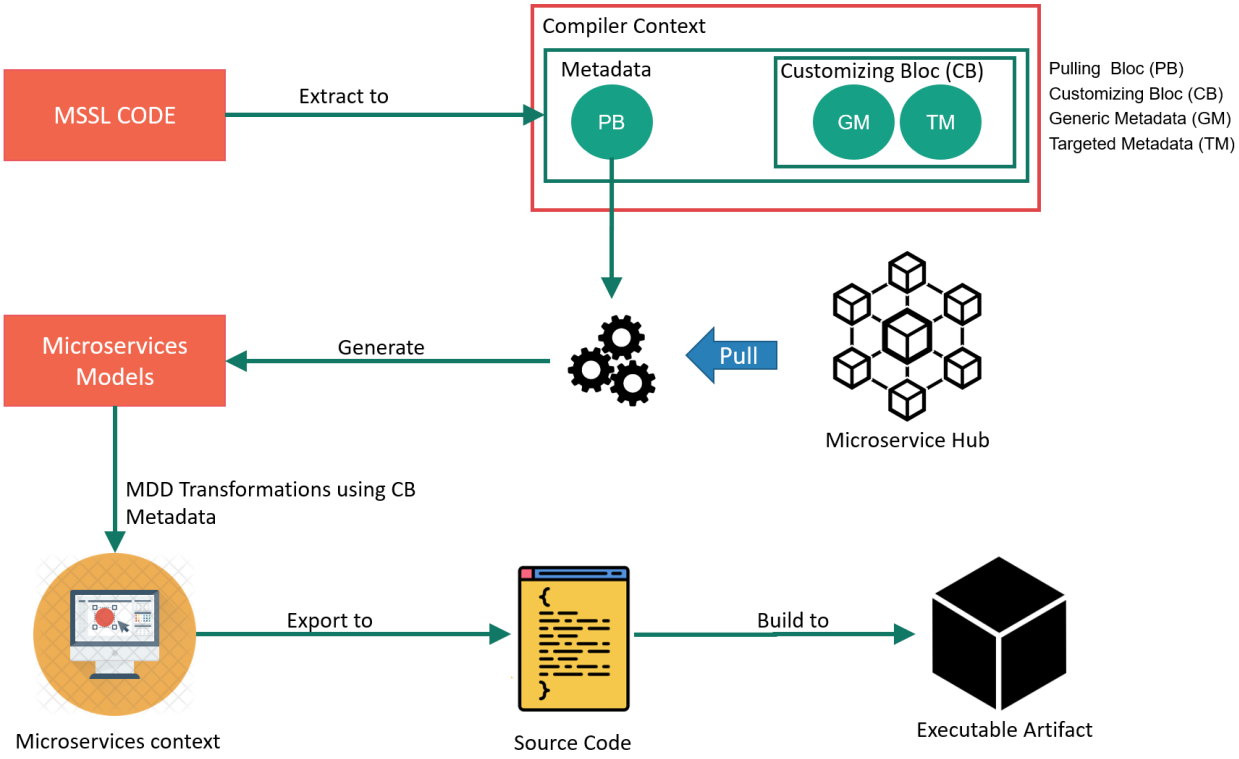


Fig. 2: The MSSL processing model

All these processing stages (export, build, etc...) are handled by a Command-Line Interface (CLI), this CLI is a tool that allows developers to interact with the MSSL by typing commands in a text-based interface. Its primary role is to provide developers with a flexible and powerful way to perform a wide range of tasks, including managing microservices, running tests, building projects or a specific microservice, managing versions, automating tasks through scripting, etc. The CLI is especially useful for advanced users and project administrators who need to perform complex tasks quickly and efficiently.

The strength of MSSL is clearly linked to the size of the MH, as we mentioned, the MSDD enables the MH to be always fed with new microservices and features. So the larger the number of new projects with new microservices or features that are created, the bigger the size of the MH will be, and the stronger the MSSL will become.

The MSSL provides an additional layer of abstraction on top of the MDD transformation. MSSL is essentially an abstraction of all microservices in the MH. For example, the development of an authentication system by MDD must start by identifying the system requirements and then create a model of the system by using a modeling language, such as UML or SysML and create models of the authentication system that captures its structure, behavior, and interactions with other systems. These models should include entities such as users, authentication tokens, access control policies, and forms (sign in, sign up . password reset, etc ), as well as the processes for authenticating and authorizing users.
In contrast, Code 1 provides us with the capability to build an authentication system using the MSSL, comprising just 13 lines of declarative code, it will generate a microservice called "Users", which enables users to sign in, sign up and reset their password using either phone or email and automatically generates their username. Furthermore, it encourages the use of strong passwords with two-factor authentication and is based on two roles: Administrator and Editor.

We have implemented MSSL based on YAML[15] Markup Language. YAML is an easy-to-read data serialization language for all programming languages (This is not mandatory, any form of descriptive language, including graphical syntax, can be used), Developers can state what they want for features, and only have to intervene in special cases when a project includes a microservice with a feature that cannot be built on existing microservices features in the MH or through generic models. If this is the case, this feature must be added to a microservice in the MH or as a new generic model. The MSSL generates the default UI/UX used in the MH's microservices, so the UI/UX must be customized manually or by a UI/UX builder.

Code 1 Authentication system.
1: UserService as Users :

[15] https://yaml.org/

```
2:  AutoGenerateUsername : true
3:  PasswordMin : 6
4:  TwoFactorAuth : true
5:  AuthPassport :
6:    - Email
7:    - Phone
8:  Roles :
9:    - Administrator :
10:       - Register: false
11:       - SuperUser: true
12:   - Editor :
13:       - Register: true
```

## 5.2 Full proposed model

Our proposed model is the full integration of the MSSL language framework with the MSDD development approach to build the MSSL-based Low-Code (MSLC) platform. The MSDD provides the tools to build and maintain the MH, while the MSSL provides the means to utilize and customize the microservices from the MH. As a result, the MSSL makes the MSDD process semi-to-fully automated and the developers can focus on the business domain requirements of the project.

Fig. 3 shows the whole process, from identifying the business requirements of the project in order to extract microservices, to deploying the project. Companies have the flexibility to choose any decomposition method, such as a domain-driven or data flow-driven approach to extract the microservices from the project. As explained previously in Algorithm 1, The extracted microservices must be checked against the MH to detect new microservices. If any new microservices are found, they must be created and pushed to the MH. Additionally, existing microservices in the MH must be checked for new features that cannot be developed from the existing models in their associated microservices or generic models. These new features must be developed and added to their associated microservices, and the upgraded microservices must then be pushed to the MH, thus completing the MSDD development process. After that, the project is described by the MSSL code, which is then processed by the MSSL compiler, as previously explained in Fig. 2. It is imperative to note that all microservices must be created using the MDD approach in order to ensure compatibility between the MSDD and MSSL, as the MSSL relies on the MDD models to customize and generate the projects source codes and executable artifacts.

Companies have the flexibility to pick any type of syntax for MSSL, including YAML, JSON, or even a visual syntax. Additionally, they can incorporate technical tools such as IDE, UI/UX

builder, CD/CI integration, and more. These tools are interchangeable, depending on the company, as long as the roles of MSSL and MSDD are preserved.

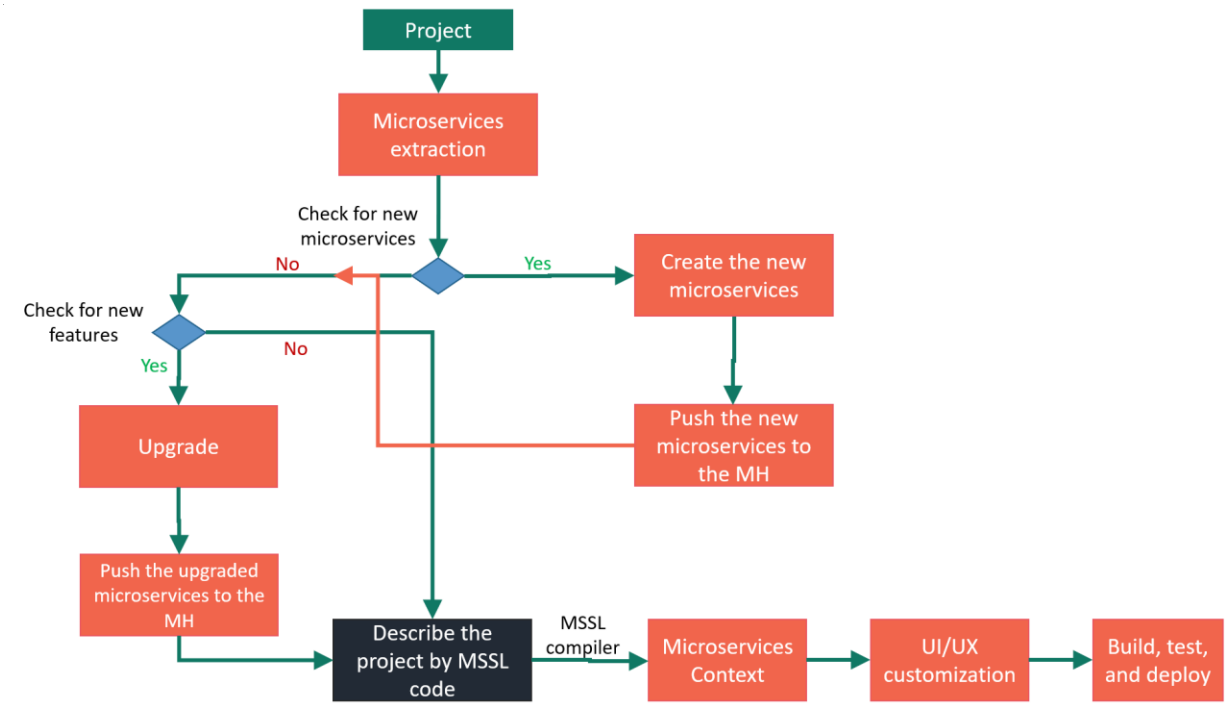


Fig. 3: Full proposed model

The Identification and classification of the MH is a crucial step to implementing our model because the necessary resources and time for project development depend on the existence of their microservices in the MH. Before adding a microservice to the MH, it must be classified by domain and its potential for cross-domain usage, in some cases, a microservice will be used just one time in one project, in these cases, this microservice is not necessary to be added to the MH, this can lead to wasting resources and time during MSSL development. The classification of microservices will identify which ones should be prioritized for development.

The size of the company and how long the company has been in the market are both factors that will determine how a company can build its own MH.

Startups and new companies can benefit from the data outlined in this paper. Certain software companies specialize in particular fields, like Education or Health, in order to begin, these startups must initially identify the global business requirements of their domain and then turn them into pre-developed microservices. With these pre-developed microservices, as well as cross-domain microservices that include user management services, payment management services, etc., startups can build a powerful MH.

Older companies can also benefit from their own projects to build an MH; their MSA-based projects can be easily integrated into the MH, but monolithic projects must be migrated to MSA.

## 6. Case Study

In this section, we will provide an overview of our MSSL capabilities by demonstrating a case study of creating a web marketplace. We developed the initial version of our MSLC platform with Node.js[16] and TypeScript[17] and, this initial version has the capability to generate code in JavaScript[18]/TypeScript for the Node.js environment, this is due to JavaScript's ability to develop the frontend and backend of microservices across all platforms and support Progressive Web Apps (PWA). We chose not to spend time creating a generator that can manage multiple languages in this initial version, so we could remain focused on validating our model. Our MSLC platform, built on top of GrapesJS[19] - an open-source graphical builder - provides UI/UX customization. The UI/UX customization can be done after the generation of the MC by the MSSL. Because the MSSL is based on prebuilt and already-used microservices, each microservice has a default UI/UX accompanying the generated source code. The MSSL IDE is built on Visual Studio Code (VSCode)[20], using a set of plugins. Because of the VSCode ecosystem flexibility, it was easy to integrate third-party libraries such as Cypress[21] for end-to-end testing, version control integration, and CI/CD Pipelines. Our MSLC also supports infrastructure-as-code at the deployment level and can generate Terraform[22] configuration files for development, testing, and deployment stages.

This case study involves the development of an online marketplace platform that connects buyers and sellers, allowing them to transact goods. The focus is placed on the MSSL due to it being the core model of the LCP, while other components such as the UI/UI builder and supporting languages could be altered depending on the company. The development of this project will involve the following core elements:

- ➢ User Interface: A user interface that allows buyers and sellers to easily navigate the platform and conduct transactions.
- ➢ Product Catalog: A product catalog that lists all items available for sale on the platform, along with relevant information such as price, images, and product descriptions.

[16] https://nodejs.org/
[17] https://www.typescriptlang.org/
[18] https://www.javascript.com/
[19] https://grapesjs.com/
[20] https://code.visualstudio.com/
[21] https://www.cypress.io/
[22] https://www.terraform.io/

- ➢ Search and Filter: A search and filter feature that allows users to find specific products or services based on their needs or preferences.
- ➢ Shopping Cart: A shopping cart that allows buyers to add items to their cart and make purchases in a single transaction.
- ➢ Payment Gateway: A secure payment gateway that allows buyers to pay for their purchases and sellers to receive payment for their sales.
- ➢ Order Management: An order management system that tracks orders and provides visibility into the shipping and delivery process.
- ➢ Reviews and Ratings: A system for buyers to leave reviews and ratings of the products or services they have purchased, helping to build trust and reputation among buyers and sellers.

This project is described by Code 2, which is composed of three sections:
Section 1 is dedicated to the metadata of the platform itself, such as the version.
In Section 2, a global configuration can be set for the entire project, this applies to all microservices, such as enabling logging and payment gateways, as well as identifying environment variables like API keys.
Section 3 describes the microservices and their relationship. In this case, we start by instantiation a microservice named Users from UserService by the word "as", allowing us to specify various parameters such as AutoGenerateUsername to auto-generate a username, PasswordMin to determine the length of the password, and TwoFactorAuth to enable two-factor authentication. Additionally, this web application allows users to log in with their email or phone number and password, as well as with Google, Facebook, and Twitter accounts. Furthermore, three roles are available for users of this application: Administrator, Seller, and Customer. An Administrator is a SuperUser that cannot register like a Seller or Customer, instead, they must be added by another Administrator. To add the first Administrator, we can use the create option in the Administrator description section (line 20) and enter the first name, last name, email, and default password, or use the built-in CLI.
By employing the same method, we instantiate a Product Microservice from ProductService. This allows us to specify who can add, update, and delete products, apart from the SuperUser, by using the CanOwnedBy option. We can add new attributes in the Attributes section (line 30). The directive @add informs MSSL that it should keep the default attributes such as media, name, price, description, etc. For example, we added a new attribute named Condition with a Select Type and two values: New and Used. The product reviews can be specified as a relationship in the Relationships section (line 36), referring to the Reviews template of FeedbackService. The templates in MSSL are a specific instance of a microservice, meaning we can use FeedbackService directly and identify the attributes of the reviews (range, comment variables) and who can manipulate these attributes. However, we can save time by using the Reviews template, which encapsulates all customizations for the Reviews microservice. Our MSSL includes many templates, such as the Comments template from FeedbackService, and Student from UserService.

There are also templates for Seller and Customer, but we cannot illustrate all the MSSL technical capabilities here. Instead, we chose this case study to provide a clear understanding of our proposed model.

The ProductService has a relationship with the Search Service, providing all searching and filtering features (e.g. name, price range, etc.). To add or customize attributes must mention them in the Indexation section (line 40), here, the @add directive can be used to add the attribute Condition to the search features. The Inventory can also be used to customize the shopping cart, with a restriction that customers cannot have a total amount over 10000 MAD (Moroccan Dirham) or 1000 USD. The Shipping option allows for configuration of the Shipping and Tracking service, with a restriction to only Morocco and cities Casablanca and Rabat for shipping.

We can customize our ProductService to create a catalog of products in two different ways. The first is to use ready-to-use customization such as Categories, the Depth property set the subcategory's depth (Line 54). For the second method, we can use the @new directive (Line 55) to create a new catalog by Tags. This requires us to specify the attributes and type of relationship; in this case, a Tag is composed of a name and slug, and a product can belong to multiple tags, while a tag can have many products.

Code 2 Project Development Process.

```
1:  %MSSL  1.2
2:  ---
3:  Global :
4:    Logging : all
5:    PaymentGateways :
6:      - Stripe
7:      - Paypal
8:  UserService as Users :
9:    AutoGenerateUsername : true
11:   PasswordMin : 6
12:   TwoFactorAuth : true
13:   AuthPassport :
14:     - Email
15:     - Phone
16:     - Google
17:     - Facebook
18:     - Twitter
19:   Roles :
20:     - Administrator :
21:       - Register: false
22:       - SuperUser: true
23:     - Seller :
```

```
      - Register: true
    - Customer :
      - Register: true
ProductService as Products:
  CanOwnedBy :
    - @Users.Roles.Seller
  Attributes @add :
    - Condition :
      - Type : Select
      - Values :
        - New
        - Used
  Relationships :
    - @FeedbackService.Reviews :
      - range : 5
      - WithComment : true
  Indexation @add :
    - Condition
  Inventory : #Shopping Cart
    - MaxTotalAmount : 10000
    - Currency : USD # Moroccan Dirham
  Shipping : #Shipping and Tracking service
    - Restricted :
      - Countries :
        - Morocco
      - Cities :
        - Casablanca
        - Rabat
  CatalogueService :
    Categories as ProductCategories:
      - Depth : Infinity
    Tags @new :
      - Attributes :
        - name :
          - Type : Text
          - MaxLength : 50
          - MinLength : 2
        - slug @slug:
          - Target : name
      - Type : ManyToMany
```

While this case study provides a detailed presentation of the MSSL's performance and effectiveness in a specific use case scenario, it is important to note that it only scratches the surface of the MSSL's full technical capabilities. As with any software application, there are multiple features and functionalities that are often not fully utilized in a single use case or scenario.
We recognize that this case study may not showcase all of the MSSL's features and capabilities. For example, it may not cover certain aspects such as the MSSL's ability to integrate with other software systems, its compatibility with different operating systems for cross-platform software development, or its ability to handle large-scale projects with maintaining and upgrading.

## 7. Evaluation

The fact that our proposed model is implemented and used in two software companies is a strong proof-of-concept because this model was developed for the first time as a solution to reduce and control the interaction with the MH after that, we have identified and isolated the key elements of the solution and create a model that can be implemented on any software company. We must just implement it in a new environment and analyze the feedback from LCP experts.

The three researchers of this work overseeing this evaluation were two Ph.D. teachers from a university specializing in computer technology, as well as a senior software architect and PhD candidate in computer science.

Our aim in this evaluation was to determine whether our model could be feasibly developed by any software company and verify if our proposed model is able to overcome the LCP challenges. To test this, we evaluated it in a small startup of 10-50 employees, which specialized in digital marketing solutions. This startup was chosen to verify if our model could be implemented in any size of companies. To have a base to compare our model, we chose this startup because it already uses MDD and LCPs in its projects.
We supplied this startup with all the data outlined in Section 4, and they have chosen to create only the microservices that are applicable to their business domain. Since the domain model details are confidential to the two companies listed in Section 4, we did not provide them to this startup.

A team of seven senior software engineers in this startup starts by identifying their domain requirements, which involves identifying the key entities, attributes, and relationships needed to represent each microservice category. From there, PIM is created, which defines the business processes in a technology-agnostic way. The next step was to generate the PSM using their own model transformation tool, which translates the PIM into a specific technology platform. With the PSM in place, they start building an instance of each microservice category, which involves code checking, testing, and fixing any bugs or issues that arise. Once a microservice is developed and tested, they add it to the MH.

The creation of the MH is the most essential element of our model. By constructing it, we can then proceed with the development of the MSSL, which is simply an abstraction of the MH microservices. Once the MSSL had been completed, it was then integrated with a UI/UX builder. The supervisors (researchers of this work) have finally tested and validated the developed platform.

We conducted a qualitative analysis by having ten software engineers and LCPs experts from the startup use the created LCP and the legacy LCPs (the original LCPs used to build the projects) to rebuild only the business requirements of their last three projects that were developed using other LCPs because the platform is composed of components that depend on each company such as UI/UX builder, we focused on the development process by the MSSL, not the whole platform, and for that, we concentrated solely on the business requirements without considering UI/UX customization. We used their last three projects for the reason that the requirements gathering had already been completed and the technical specification already detailed, thus, negating their impact.

Each of the selected experts has at least five years of experience in LCPs and more than ten years of experience in software engineering, they possess the expertise and knowledge to evaluate our model and all of them are familiar with MDD and MSA in order to prevent MDD and MSA from impacting the evaluation of the MSLC.

Each engineer was interviewed by script with four questions (Q1-Q4) :
Q1: Are there any disconcerting or uncomfortable aspects to utilizing the MSLC platform?
Q2: What key distinctions exist between the MSLC platform and the platforms you have previously utilized?
Q3: Would you prefer to keep utilizing this platform over any other alternatives?
Q4: Evaluate the capability of the MSLC platform to address the seven LCP challenges (Table 1).

## 7.1 Results

All projects were successfully completed by all experts. Table 3 shows the time spent learning, and on each project (P1, P2, P3) for each expert, while the orange background represents the time spent using the legacy LCP, the green background is for the time spent using the MSLC platform. The average time that the experts spent during the entire study were 32h07, on average, it took 6h48 to learn, with the rest of the time being devoted to the three projects. We have to mention that if the selected experts are not familiar with MDD and MSA, then it might take more than that. The stakeholders in the startup where the evaluation is being conducted accept this learning average very well.

During the project's rebuild, the experts face the problem that the legacy LCPs are all built to support the UI/UX first, which doesn't allow for the separation of business requirements from the UI/UX. In contrast, the MSLC platform allows the UI/UX to be just one step of the development process, and the entire project can be built without spending time on the UI/UX due to the prebuilt microservices in the MH. This explains why the average time spent developing on the legacy LCP (6h42) is so much greater than that on the MSLC platform (1h43). This is a great advantage for business softwares and commercial softwares, as most business software does not require a fancy UI/UX as much as they need a stable graphical user interface.

| Exp. | Learn. | P1 | P2 | P3 | P1 | P2 | P3 | Total |
|---|---|---|---|---|---|---|---|---|
| 1 | 7h00 | 5h00 | 6h30 | 10h30 | 2h00 | 1h00 | 3h15 | 35h15 |
| 2 | 9h00 | 7h30 | 6h45 | 4h30 | 1h30 | 1h15 | 1h30 | 32h00 |
| 3 | 6h00 | 8h15 | 4h00 | 9h15 | 1h15 | 1h30 | 2h30 | 32h45 |
| 4 | 7h00 | 9h00 | 7h30 | 9h00 | 2h00 | 2h00 | 2h15 | 38h45 |
| 5 | 6h00 | 4h30 | 6h15 | 8h45 | 00h45 | 1h45 | 2h00 | 30h00 |
| 6 | 5h00 | 6h00 | 4h15 | 5h00 | 2h00 | 1h00 | 1h15 | 24h30 |
| 7 | 7h00 | 6h45 | 8h00 | 4h30 | 2h45 | 2h30 | 0h45 | 32h15 |
| 8 | 7h00 | 5h15 | 5h30 | 8h30 | 1h30 | 1h15 | 2h45 | 31h45 |
| 9 | 8h00 | 9h30 | 6h00 | 3h00 | 2h15 | 2h00 | 0h45 | 31h30 |
| 10 | 6h00 | 7h00 | 8h30 | 6h15 | 1h45 | 1h30 | 1h30 | 32h30 |
| Ave. | 6h48 | 6h52 | 6h19 | 6h55 | 1h46 | 1h34 | 1h51 | 32h07 |
| | | Ave : 6h42 | | | Ave : 1h43 | | | |

Table 3: Data obtained from Q1 and Q2

Q1: Seven of ten (70%) of the experts expressed worry about potentially wasting time when extracting microservices from software requirements, whereas three of ten (30%) of them expressed unease about code-first, not UI/UX-first, as it differs from other platforms they are accustomed to, the UI/UX in MSLC is just a step, not the whole process.
Q2: All the experts agree that flexibility is the major advantage of the MSLC. They stated that they do not feel confined to one platform and that the ability to access the code level at any time is reassuring, this is unlike other LCPs, and some even require you to utilize their cloud service.

Q3: Two of ten (20%) experts prefer to only use the MSLC platform on large projects, such as ERPs and legacy LCPs, while small projects, such as e-commerce web applications, are excluded. However, eight of ten (80%) experts intend to use the MSLC platform in all their next projects.
Q4: Table 4 illustrates the evaluation of the ten experts on the capacity of our model to tackle the seven LCP challenges. Our model achieved an average rating of 9.17/10. It was assessed as more than 9/10 for CH1, CH2, CH3, CH4, and CH7, with some concerns on CH5 and CH6, where it was rated 8.1/10 and 8/10 respectively. Our model relies on both MSA and MDD, so the MSLC developers must be knowledgeable in MSA in order to use and maintain it. MDD is not a necessity for the developers users, except when upgrading the MH.

| Exp. | CH1 | CH2 | CH3 | CH4 | CH5 | CH6 | CH7 |
|---|---|---|---|---|---|---|---|
| 1 | 10/10 | 9/10 | 10/10 | 8/10 | 9/10 | 7/10 | 9/10 |
| 2 | 9/10 | 10/10 | 10/10 | 10/10 | 7/10 | 8/10 | 9/10 |
| 3 | 10/10 | 10/10 | 10/10 | 10/10 | 8/10 | 9/10 | 10/10 |
| 4 | 10/10 | 10/10 | 10/10 | 9/10 | 9/10 | 9/10 | 10/10 |
| 5 | 9/10 | 10/10 | 10/10 | 10/10 | 8/10 | 9/10 | 10/10 |
| 6 | 8/10 | 9/10 | 10/10 | 8/10 | 6/10 | 7/10 | 9/10 |
| 7 | 10/10 | 8/10 | 10/10 | 10/10 | 8/10 | 8/10 | 10/10 |
| 8 | 9/10 | 10/10 | 10/10 | 10/10 | 8/10 | 7/10 | 9/10 |
| 9 | 10/10 | 10/10 | 10/10 | 9/10 | 9/10 | 8/10 | 10/10 |
| 10 | 10/10 | 10/10 | 10/10 | 10/10 | 9/10 | 8/10 | 10/10 |
| Ave. | 9.5/10 | 9.6/10 | 10/10 | 9.4/10 | 8.1/10 | 8/10 | 9.6/10 |
| Total. | 9.17/10 | | | | | | |

Table 4: The expert's evaluation

## 8. Discussion

The only drawback of our model is that it requires an MDD and MSA environment to be set up, which may impede its ability to handle CH5 and CH6. Fortunately, these two concepts have gained much popularity in recent years, especially MSA, which is now obligatory in many software companies. The use of microservices architectures has been gaining traction due to its many advantages, such as faster time-to-market, scalability, and enhanced response times. These features

have motivated stakeholders to become increasingly involved in the development and deployment of microservices, allowing them to bring their business vision to life [31].
Our model was created with code-first in mind; it provides access to the source code throughout all stages of software development, giving developers full control over their projects. This surpasses CH1, CH3, and CH4, enabling developers to directly address security issues and add new features to the source code without the need to wait for the upgrade or update of the LCP itself. Additionally, our model benefits from the MSA mechanisms to address security concerns. Firstly, each microservice has a well-defined boundary and interface, which limits its attack surface. Therefore, if a security vulnerability is found in one microservice, it will not necessarily affect the entire system. Secondly, microservices can be deployed in their own containers, providing an additional layer of isolation and security. This means that even if one microservice is compromised, it will not necessarily affect other microservices in the system. Thirdly, MSA enables fine-grained authentication and authorization, allowing each microservice to control access to its own resources. This helps to prevent unauthorized access to sensitive data and functions. Fourthly, microservices can use encryption to protect data in transit and at rest, ensuring that data is not accessible to unauthorized parties. Lastly, microservices can be designed to log all interactions and events, which can be used for auditing and monitoring purposes. This helps to detect and respond to security breaches in real-time [27] [28].

The MSA capability to reduce integration complexity is also present in this model, which helps to surpass CH2 by decoupling components, enabling scalability, supporting technology diversity, and allowing for continuous integration and deployment. Firstly, microservices are designed to be modular and independent, with clear boundaries and interfaces. This means that each microservice can be developed, tested, and deployed independently of other microservices, reducing the need for complex integrations between different components. Secondly, microservices can be scaled independently based on demand, eliminating the need for complex integrations that would be required to scale a monolithic application. Thirdly, MSA allows for different technologies to be used for different microservices, removing the need for complex integrations between components that use different technologies. Lastly, microservices can be developed, tested, and deployed independently using CI/CD pipelines, which helps to reduce integration complexity by automating the integration and deployment process and ensuring that each microservice is tested and deployed in isolation.

MSA also addresses the difficulty of maintaining and repairing Low-Code based software. it encourages the decoupling of various system components into individual services that can be developed, deployed, and maintained independently. This reduces the likelihood of a small change to one service causing issues in other parts of the system. MSA allows for easy scalability of individual services, rather than having to scale the entire monolithic application, this means that the system can adapt to changing demand and usage patterns without affecting the performance of other services. With MSA, it is easier to recover from failures in individual services. If a service

goes down, the rest of the system can continue to function, and the affected service can be replaced or repaired without affecting the other services. MSA makes it easier to test individual services in isolation, which can reduce the likelihood of bugs and issues being introduced into the system. Finally, with MSA, it is easier to monitor the performance of individual services and quickly identify any issues that arise. This allows for proactive maintenance and repair of the system, rather than waiting for a larger issue to arise.

## 9. Conclusion and future work

This paper presents a novel LCP development model, which is a result of the adoption of the MSDD approach in two software companies over three years. This model provides software companies with a strong foundation to build their own LCPs, using an MSSL for a declarative integration of the MSA into LCPs. To validate our model, it was implemented in a third software company and evaluated by ten LCP experts. The results of this evaluation confirmed that our model can effectively overcome five LCPs challenges. which are the customization limitations, integration complexity, vendor lock-in, security concerns, and the difficulty of maintaining LCPs. With regards to technical debt and learning curve, these can be surpassed if the implementation environment is familiar with MDD and MSA.

We've likened our model to the Docker Hub working method, pushing, pulling, and customizing from the MH, and we proved that the bigger the MH, the more powerful our model. Therefore, in our future work, we plan to create a standard, open-source version of MSSL using YAML and construct an online Hub for microservices that can be shared among companies and stakeholders. This way, every stakeholder can work to contribute to expanding the MH.
We plan to provide comprehensive online documentation that outlines all of the MSSL's features and functionalities. This documentation will provide a thorough explanation of the MSSL's capabilities and how they can be leveraged in various use cases and scenarios. Additionally, we will provide examples of how the MSSL has been used in various contexts, highlighting its versatility and adaptability. While the presented case study provides valuable insights into the MSSL's performance in a specific use case, it is important to recognize that it is just one part of a larger picture. Our goal is to provide users with a complete understanding of our tool's technical capabilities, and we believe that the online documentation will be a valuable resource in achieving this.

## Acknowledgment

The research work reported in this paper received financial support from Center for Doctoral Studies grants - Hassan First University - Morocco.

## References

1: Number of mobile devices worldwide 2020-2025 | Statista. (2023, March 10). Statista. https://www.statista.com/statistics/245501/multiple-mobile-device-ownership-worldwide/
2; Global IoT and non-IoT connections 2010-2025 | Statista. (2022, September 6). Statista. https://www.statista.com/statistics/1101442/iot-number-of-connected-devices-worldwide/
3; IBISWorld - Industry Market Research, Reports, and Statistics. (n.d.). https://www.ibisworld.com/industry-statistics/number-of-businesses/point-of-sale-software-developers-united-states/
4: Lewis, J., & Martin, F.: Microservices. A definition of this new architectural term. Retrieved from MartinFowler.com (2014): https://martinfowler.com/articles/microservices.html
5: Di Francesco, P., Malavolta, I., & Lago, P. (2017). Research on Architecting Microservices: Trends, Focus, and Potential for Industrial Adoption. 2017 IEEE International Conference on Software Architecture (ICSA). https://doi.org/10.1109/icsa.2017.24
6: Zimmermann, O. (2017). Microservices tenets. Computer Science - Research and Development, 32(3–4), 301–310. https://doi.org/10.1007/s00450-016-0337-0
7: N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina, "Microservices: yesterday, today, and tomorrow," Present and ulterior software engineering, pp. 195–216, 2017.
8: Newman, S. (2015). Building Microservices. Oreilly & Associates Incorporated.
9: Hasselbring, W., & Steinacker, G. (2017). Microservice Architectures for Scalability, Agility and Reliability in E-Commerce. IEEE International Conference on Software Architecture Workshops. https://doi.org/10.1109/icsaw.2017.11
10 : Gos, K., & Zabierowski, W. (2020). The Comparison of Microservice and Monolithic Architecture. International Conference on Perspective Technologies and Methods in MEMS Design. https://doi.org/10.1109/memstech49584.2020.9109514
11: Tapia, F.; Mora, M.Á.; Fuertes, W.; Aules, H.; Flores, E.; Toulkeridis, T. From Monolithic Systems to Microservices: A Comparative Study of Performance. Appl. Sci. 2020, 10, 5797. https://doi.org/10.3390/app10175797
12: Soylemez, M. T., Tekinerdogan, B., & Tarhan, A. (2022). Challenges and Solution Directions of Microservice Architectures: A Systematic Literature Review. Applied Sciences, 12(11), 5507. https://doi.org/10.3390/app12115507
13: J. Ghofrani and D. L¨ubke, "Challenges of microservices architecture: A survey on the state of the practice." in ZEUS, 2018, pp. 1–8.
14: Viggiato, M., Terra, R., Rocha, H., Valente, M.T., Figueiredo, E.: Microservices in practice: a survey study. CoRR (2018).
arXiv:1808.04836
15:The Forrester WaveTM: Low-Code Development Platforms For. . . | Forrester. (n.d.). Forrester. https://www.forrester.com/report/The-Forrester-Wave-LowCode-Development-Platforms-For-Professional-Developers-Q2-2021/RES161668

16 : Mendix. (2023, March 20). Gartner Magic Quadrant for Enterprise Low Code Application Platforms 2022. https://www.mendix.com/resources/gartner-magic-quadrant-for-low-code-application-platforms/
17: Mendix. (2023a, January 25). What is Model-Driven Development (MDD)? Tools, Benefits & FAQ. https://www.mendix.com/model-driven-development/
18: New Development Platforms Emerge For Customer-Facing. . . | Forrester. (n.d.). Forrester. https://www.forrester.com/report/New-Development-Platforms-Emerge-For-CustomerFacing-Applications/RES113411
19: Gomes, P. M., & Brito, M. A. (2022). Low-Code Development Platforms: A Descriptive Study. 2022 17th Iberian Conference on Information Systems and Technologies (CISTI). https://doi.org/10.23919/cisti54924.2022.9820354
20: Juhás, G., Molnar, L., Juhasova, A., Ondrisova, M., Mladoniczky, M., & Kovacik, T. (2022). Low-code platforms and languages: the future of software development. 2022 20th International Conference on Emerging ELearning Technologies and Applications (ICETA). https://doi.org/10.1109/iceta57911.2022.9974697
21: Alamin, A. A., Malakar, S., Uddin, G., Afroz, S., Haider, T. B., & Iqbal, A. (2021). An Empirical Study of Developer Discussions on Low-Code Software Development Challenges. ArXiv (Cornell University). https://doi.org/10.1109/msr52588.2021.00018
22: Rademacher, F.; Sorgalla, J.; Wizenty, P.N.; Sachweh, S.; Zündorf, A. Microservice architecture and model-driven development: Yet singles, soon married (?). In Proceedings of the 19th International Conference on Agile Software Development: Companion, Porto, Portugal, 21–25 May 2018; pp. 1–5.
23 : Aksakalli, I.K.; Celik, T.; Can, A.B.; Tekinerdogan, B. A Model-Driven Architecture for Automated Deployment of Microservices. Appl. Sci. 2021, 11, 9617. https://doi.org/10.3390/app11209617
24 : Vernon, V. (2016). Domain-driven Design Distilled. Addison-Wesley Professional
25 : Li, S., Zhang, H., Jia, Z., Li, Z., Zhang, C., Li, J., Gao, Q., Ge, J., & Shan, Z. (2019). A dataflow-driven approach to identifying microservices from monolithic applications. Journal of Systems and Software, 157, 110380. https://doi.org/10.1016/j.jss.2019.07.008
26 : P. Merson and J. Yoder, "Modeling Microservices with DDD," 2020 IEEE International Conference on Software Architecture Companion (ICSA-C), Salvador, Brazil, 2020, pp. 7-8, doi: 10.1109/ICSA-C50368.2020.00010.
27: Ali, R. N., Mojtaba, S., Raviz, S. a. H., Liang, P., Mashmool, A., & Valentina, L. (2022b). An empirical study of security practices for microservices systems. Journal of Systems and Software, 198, 111563. https://doi.org/10.1016/j.jss.2022.111563
28: Mateus-Coelho, N., Cruz-Cunha, M., & De Souza Ferreira, L. C. (2021). Security in Microservices Architectures. Procedia Computer Science, 181, 1225–1236. https://doi.org/10.1016/j.procs.2021.01.320
29:X. Larrucea, I. Santamaria, R. Colomo-Palacios and C. Ebert, "Microservices," in IEEE Software, vol. 35, no. 3, pp. 96-100, May/June 2018, doi: 10.1109/MS.2018.2141030.

30: Nadareishvili, I.; Mitra, R.; McLarty, M.; Amundsen, M. Microservice Architecture: Aligning Principles, Practices, and Culture; O'Reilly Media Publishing: Sebastopol, CA, USA, 2016.
31: Márquez, G., Taramasco, C., Astudillo, H., Zalc, V., & Istrate, D. (2021). Involving Stakeholders in the Implementation of Microservice-Based Systems: A Case Study in an Ambient-Assisted Living System. IEEE Access, 9, 9411-9428.
32: Pefers K, Tuunanen T, Rothenberger MA, Chatterjee S (2007) A design science research methodology for information systems research. J Manag Inf Syst 24(3):45–77
33: March, S. T., & Smith, G. R. (1995). Design and natural science research on information technology. Decision Support Systems, 15(4), 251–266. https://doi.org/10.1016/0167-9236(94)00041-2
34: Mendix. (2023c, March 20). Gartner Magic Quadrant for Enterprise Low Code Application Platforms 2022. https://www.mendix.com/resources/gartner-magic-quadrant-for-low-code-application-platforms/
35: Mendix. (2023a, January 17). Gartner Magic Quadrant & Forrester Wave Analyses | Mendix Evaluation Guide. https://www.mendix.com/evaluation-guide/gartner-forrester-mendix/