applied sciences

MDPI

*Article*

# MSDeveloper: A Variability-Guided Methodology for Microservice-Based Development

**Betul Kuruoglu Dolu [1,2,*], Anil Cetinkaya [2,3], M. Cagri Kaya [2,4], Selma Nazlioglu [5] and Ali H. Dogru [6]**

1 ASELSAN, Ankara 06200, Türkiye
2 Department of Computer Engineering, Middle East Technical University, Ankara 06800, Türkiye
3 Department of Computer Engineering, Iskenderun Technical University (ISTE), İskenderun 31200, Türkiye
4 Department of Computer Engineering, Ardahan University, Ardahan 75002, Türkiye
5 Department of Software Engineering, Atilim University, Ankara 06830, Türkiye
6 Department of Computer Science, The University of Texas at San Antonio, San Antonio, TX 78249, USA
* Correspondence: bkdolu@aselsan.com.tr or betul.kuruoglu@ceng.metu.edu.tr

**Abstract:** This article presents a microservice-based development approach, MSDeveloper (Microservices Developer), employing variability management for product configuration through a low-code development environment. The purpose of this approach is to offer a general-purpose environment for the easier development of families of products for different domains: a domain-oriented development environment is suggested, where domain developers and product developers can utilize the environment as a software ecosystem. Thus, genericity is offered through supporting different domains. A domain is populated with feature and process models and microservices in a layered architecture. Feature models drive the product configuration, which affects the process model and the microservice layer. An experimental study was conducted to validate the applicability of the approach and the usability of the development environment. Students from different courses were assigned system modeling projects where they utilized helper tools supporting the provided methodology. Furthermore, professional software developers were consulted about this recommended domain-oriented development environment. Feedback from student projects and professionals' remarks are analyzed and discussed.

**Keywords:** low-code development; microservices architecture; model-driven engineering; software development; variability modeling; process modeling



**Citation:** Kuruoglu Dolu, B.; Cetinkaya, A.; Kaya, M.C.; Nazlioglu, S.; Dogru, A.H. MSDeveloper: A Variability-Guided Methodology for Microservice-Based Development. *Appl. Sci.* **2022**, *12*, 11439. https://doi.org/10.3390/app122211439

Academic Editors: Sanjay Misra, Robertas Damaševičius and Bharti Suri

Received: 18 October 2022
Accepted: 8 November 2022
Published: 11 November 2022



## 1. Introduction

With the changing world, software is rapidly being included in every aspect of our lives in an indispensable way. Therefore, low-cost, consistent, error-free, and user-friendly software production has become very important. Low-code development intends to serve these purposes and allows the fast development of products supported by visual modeling that will achieve needs by writing little to no code [1]. It is a development approach that responds quickly to customers' needs with reduced costs. However, most of the existing studies in this field are tool-centric, references are scarce [2] and less inclusive about conceptual frameworks, and are methodologies especially lacking.

In the world of software development, which transforms, develops, and evolves every day, software development approaches are also being improved, renewed, and expanded. With a background of almost 10 years, low-code development is a software development approach that has become quite popular. Its practical use is supported by visual modeling, which allows quick development, especially leveraging on mature domains with low costs. Low-code development environments are easy to use in a product-oriented way, even for users without a programming background, which relieves them from mundane "infrastructural" tasks. These environments are being further established in the IT sector day by day [2].

The expectations sound ambitious. We therefore impose some restrictions. First, the developers who will maintain coding phases in low-code development environments should be advanced computer users—preferably developers. Then, the development environments should be domain-oriented. This requirement relieves the developers from many development issues that are "infrastructural". For any application field, the development domain should be prepared with inclusive models and executable code for the common functions in that field. We refer to such a domain as mature [3] if there is no unwritten code left for the functions required in that field. Such code could take the form of components or microservices. This expectation is not unrealistic for today.

There are many no-code and low-code development environments that are gathering bigger developer communities. Owing to the long past of "faster development" initiatives, there are some common patterns of use and feel for such tools. Earlier Integrated Development Environments (IDE) offered drag-and-drop facilities for the "low-code" development of graphical user interfaces (GUI). Later, the "widgets" used in those environments were made database-aware. Additionally, earlier desktop database management systems were supported with such environments to quickly configure applications with GUI and databases glued with some procedural code and SQL.

Some enterprises such as Creatio (formerly bpm'online) [4], Pega [5], Salesforce [6], and Zoho [7] specialised in Customer Relationship Management (CRM) solutions, and incorporated low-code development extensions within their products via the orchestration of processes in their business process model functionalities [8]. There are also low-code tools prepared for a specific domain, such as IoT, as researched in ref. [9].

The influence of such earlier successful tools can be observed in modern approaches. We are, however, seeking a more systematic approach for related methodologies and supporting tools [10–12]. The existing tools are usually preferred for specific domains. If a more generic approach was desired, software ecosystems could probably be used in the definition of various domains, making possible the configuration of products in such domains. For genericity, the infrastructure for development should allow the definition of different domains (domain engineering), as well as product development. Additionally, a simple high-level reference architecture would benefit genericity. An ecosystem that allows the easy definition and instantiation of new domains and products will support continued infrastructural success and gain wider user communities.

Our vision is to provide a generic approach utilizing the existing experience in many architectural avenues, such as Service-Oriented Architecture (SoA) and Software Product Lines (SPL). A more foundational concern has been voiced in the past when programming languages were being discussed for offering generic capabilities: the Böhm Jacopini Theorem [13] has proven genericity through the minimum requirements for the control structures to be included in the languages for that matter. They demonstrated the Turing Machine equivalency. In a similar fashion, fundamental design dimensions in this research are intended to be addressed for genericity. These dimensions are catered by the constituents of the modern development approaches organized in our methodology.

This article reports on our ongoing work, which offers graphical modeling environments for an example application domain (Webinar System) and supports the development of products without code writing once the developers refine their set of graphical models (Feature, Variability, and Business Process models) and Microservice Domain Model in conformance with each other. A two-layer architecture is proposed for accommodating these models, corresponding to the run-time environment.

The proposed methodology offers a generic environment, which is suitable for defining different domains, allowing the development of products in a variability-guided manner. For this approach, variability stands out as the ultimate development concept. Furthermore, SPLE is expanded to low-code directions, and the expansion bridges these two concepts.

This article is organized as follows: In Section 2, the background information that forms the basis of our study is presented. In Section 3, studies regarding low-code development platforms and the modeling approaches of these studies are presented. In Section 4,

our approach and architectural designs are explained in detail. Section 5 presents our experimental study conducted for the Webinar Systems domain, the survey we conducted after the study, and professionals' remarks about our approach. In Section 6, the analysis we conducted on the results of the survey is explained. In Section 7, the general applicability and validity of the approach is discussed in addition to our concluding remarks.

## 2. Background

SoA has been a widely accepted and employed avenue for developing complex and distributed applications that usually connect through the Internet. The integration mechanisms supported through process models that are intuitive and graphical actually present an invaluable approach to development. They leverage the world market of web services that provide any kind of functionality and that are already written, tested, and published. The need to create more flexible and scalable systems leads to moving attention from developing monolithic applications to independently functioning microservices. With each managing their own data storage, microservices, which are potentially written in different languages, use lightweight protocols, such as HTTP and REST. As the focus is independent deployment, scaling, and testing, microservices require a bare minimum centralized management [14,15].

Our previous work focused mainly on the structure dimension [16], which actually corresponds to the decomposition view of Software Architecture approaches. We later realized that our research neglected an analysis of an important issue—dynamic modeling.This need has been addressed in UML through sequence diagrams or activity diagrams and in SoA by the process model. Another very powerful concept offered by the SPL approaches, namely the domain conceptions presented through feature models and variability, has also been inspirational.

SPL constitutes an environment for the construction of a set of software systems from core assets by feature management, thus making it easier to satisfy specific requirements. Common assets lie at the core of the development; therefore, the handling of these assets is crucial, in addition to variability management in the creation of a final product. Ref. [17] specified a requirement for a framework with two distinct processes: domain engineering and application engineering, with the first one being for the characterization and realization of necessary assets and the latter for the production of distinctive applications under the guidance of variability. A final product is created as a result of systematic decisions on variability resolution throughout the development phases. Each design decision corresponds to a variation point by incorporating/eliminating features. Delayed design decisions improve the efficiency of SPL by allowing the core assets to be used along with changing requirements [18]. This can be supported through variability, especially with later resolution times.

Some early approaches have been used for utilizing domain feature models in order to resolve variability; however, they were complicated because variability was accommodated in an already populated feature model. The Orthogonal Variability Model (OVM) [17] and Covamof [19] are some of the pioneering variability models that stand alone outside of feature models.

In an effort to orchestrate these concepts in the environments to further develop complex software more easily, some research was conducted regarding methodology. The decomposition of the structural view governing the components or the web services was later supported by configuring a process model for the integration and flow-control specifications. The latter addition enabled the ordering of method invocations to complete the definition of executable systems. Although today no development process should dictate a linear order of tasks, such as that of Waterfall, some priority for the modeling dimensions that would correspond to the abstraction levels of design was necessary. To support our expectations, the limited studies conducted in Ref. [20] showed that the process model can have a priority over the component model, as has practically been the case in

the SoA world, in which an implied two-level architecture is the hidden de facto standard: A process model at the top commands the web services at the bottom.

The desire to support many different ecosystem domains through potentially different communities for both defining and utilization (through no/low-code development) necessitates us to have command over the domain conceptions. That is why SPL was exploited for feature models and variability, ripening our environment. Variability will be the sole first-class citizen, probably accounting for the vast majority of the specification considerations.For this reason, we tried to align all the models under variability and support the microservices with domain-specific connector utilities [10] that play the final roles in configuring with respect to variability.

## 3. Related Work

In this section, we present some work directly related with no- and low-code software development, as well as some related research. BPMN and SPL fields are considered especially related in our work. Additionally related is the platform concept, which can be offered as a cloud service to support ecosystem capabilities.

Increasing demand in the software field has led to a never-ending search for the next development methodology. Increasing demand for complex systems composed of heterogeneous components increased the requirements for reuse and automation to meet requirements. Low-code development is a fairly new topic, and according to reports published by Gartner [8] and Forrester [21], it is expected to gain significant popularity over the next few years. The core principles are adopted from Model-Driven Engineering (MDE) approaches [22]. Waszkowski presents Aurea BPM that employs BPMN [1]. The target is to provide automation solutions for manufacturing. Model-driven software development, rapid application development, automatic code generation, and visual programming are presented as approaches that can lead to the creation of low-code programming.

A conceptual comparative framework was proposed, and a technical survey is presented in Ref. [2]. Eight low-code development platforms (LCDP) are considered as market leaders based on Refs. [8,21]: Appian [23], Google App Maker (this service has been shut down since 15 April 2020), Kissflow [24], Mendix [25], MS Power Apps [26], Salesforce Platform [6], Outsystems [27], and Zoho Creator [7]. All of the overviewed LCDPs utilized a process designer and supported workflow management in their operations. However, only Kissflow, Salesforce App Cloud, and Zoho Creator allow users to configure workflows according to their needs.

## 4. MSDeveloper: A Variability-Guided Methodology

The proposed methodology, MSDeveloper supports the development of software systems in a top-down approach. This approach is suggested with a layered model, where the process model is the model at the top layer and the structural model is at the bottom layer [20]. Both process and structural models are compliant with the variability model. The process model located at the higher level and the web services at the bottom level point to the de facto architecture in SoA. Additionally, conventional development based on UML starts with the functional dimension modeled with the use case diagrams. These widely practiced approaches, along with our research, suggest a model hierarchy where processes are at the top [20]. Rather than organizing our architecture based on structural components, such as GUI and data, etc., we prefer to process the top-level model, followed by the low-level executional models, which are microservices. These lower-level units also accommodate data structures to account for the data dimension. Following our discussion in the introduction related to genericity, fundamental design dimensions are considered in the shaping of the proposed reference architecture in Figure 1.

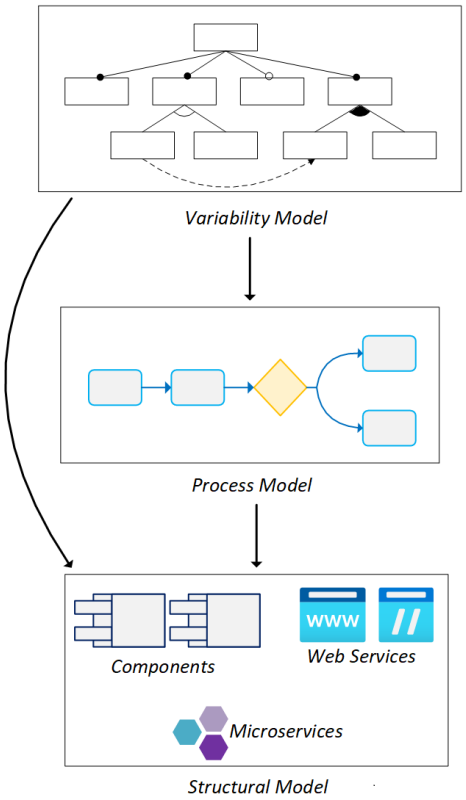


**Figure 1.** The Reference Model of MSDeveloper.

The modeling of function, data, and control support executability (Turing Machine equivalency) concerns, whereas structure supports understandability, hence the manageability of the development. These could be referred to as "3+1 dimensions of design". Table 1 lists the mapping of those dimensions to development structures. Furthermore, the process assumes control responsibilities at a higher (coordination) level. Since we are not interested in coding-level control issues, such as inside the methods, we have a better support in this dimension through process modeling.

**Table 1.** Mapping of dimensions to development structures.

| Design Dimension | Architecture Element |
|---|---|
| Control | Process model (BPMN) |
| Function | Microservice methods |
| Data | Microservice attributes |
| Structure | Two-layered architecture |

It should be noted here that the microservice philosophy suggests a single responsibility per service, practically resulting in simpler and smaller services. The orchestration assumes more duties because services are simpler. Consequently, the reference architecture does not allocate complex structures for the data and the function dimensions.

The suggested method requires a mature domain to be applied. Reusing existing code with a new configuration or generating new code from graphical models makes mature domains [3] possible, in which all common functionality and data were created earlier by domain experts. A software ecosystem can be created that supports different domains. In the suggested method, a domain in which applications will be created is assumed to be mature.Each domain in the ecosystem includes its own template process model for future applications. This step corresponds to the domain engineering stage of Software Product Line Engineering (SPLE). Additionally, developers will be able to use pre-implemented

components in their designs or create their own based on their preferences. This step corresponds to the application engineering stage of the SPLE. Based on the selected domain, users will be presented with a feature model, in addition to a variability and a process model that correspond to the feature model.

Users will be selecting the features that will be included in the final product from the variability model. Based on these selections, the process model and its required microservices will be automatically configured to incorporate the selected features.

The emphasis here is on product engineering. Preparing a domain in the ecosystem consists of the activities given in Table 2, where the order is not imposed. It is suggested to start with the domain feature model and a variability model. Rather than taking one model at a time in a linear process, all models can be developed simultaneously. Simultaneous top-down development can be supported by the Axiomatic Design [28] approach for the development of complex domains.

In a similar fashion, the product development is also suggested to start top-down and continue with a free-ordered process to modify any model any time. "Acquire Any Missing Services" activity in the product development steps will be required in the case of a non-mature domain.

**Table 2.** Domain and product development steps.

| Domain Development Steps | Product Development Steps |
| --- | --- |
| 1. Develop feature model; | 1. Resolve variabilities; |
| 2. Develop variability model; | 2. Instantiate the process model; |
| 3. Develop base process; | 3. Instantiate the microservices model; |
| 4. Develop optional processes; | a. Acquire any missing services. |
| 5. Acquire microservices. | |

*Methodological Principles*

In our work, we propose a methodology that can be used as a low-code development environment for mature domains, where the success largely depends on the maturity of domain models. The experience provided in SPL practices relieves us from the redundant declarations and definitions that took place in previous products. These are the fruits of domain orientation. Now, the low-code developers can be offered variations corresponding to a world of known products rather than the definition of requirements from scratch, which are often abstract.

There are four architectural units in MSDeveloper that are mostly defined in the domain engineering that takes place before the low-code development. Then, these models are used for the development of new applications. These units are the feature, variability, process, and microservice models. Additionally, there is a fifth unit, which was created during low-code product development: Graphical User Interface (GUI). This research excluded the GUI constituent because it has been previously addressed in depth by many applications and will be incorporated to our framework later. The framework supports the creation of different domains, and the presence of communities that can define different domains will enable the environment to support low-code development as a general approach for application development. We often use the word domain to define the set of four models that are defined for a specific application field. Figure 2 depicts the architectural components and their usage in MSDeveloper.

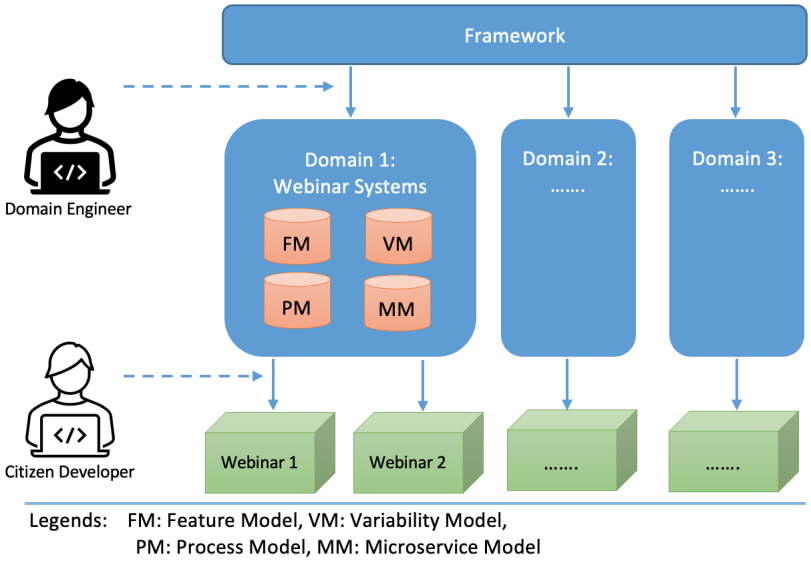


**Figure 2.** Usage of architectural components in MSDeveloper.

The transition from feature-model layer to process-model layer and the transition from the process-model layer to the microservice layer are explained below.

Domain engineering efforts are presented below, and are organized with respect to the corresponding models:

- A variability model (VM) is created (if necessary, from a feature model), as shown in Figure 3. The variability model is prepared according to the optional features and constraints that are defined in the feature model. All mandatory features that are included in every product are represented in the base feature model;
- A complete process model (PM) is prepared with Business Process Modeling Notation (BPMN), which will include every feature. This process model corresponds to the domain. The tasks, gateways, and sequence flows associated with each optionally defined feature are extracted from this main BPMN and kept as a partial BPMN for this optional feature. This operation is performed for all optional features. As a result, base (covering all mandatory features) and partial BPMNs for all optional features are prepared. Later, for the low-code development of the product corresponding to the product engineering part, according to the optional feature selections made in the variability model, the relevant tasks, gateways, and sequence flows of selected optional features are added to the base BPMN model. As a result, a product-specific BPMN model will be instantiated according to the changes made to the variability model;
- A microservice model (MM) layer is prepared to cover all the related features. A microservice may be required to be developed for each feature (some features may only correspond to processes). While the microservices prepared for mandatory features create a base microservice model, the microservices prepared for each optional feature create optional microservices. According to the optional feature selections made in the variability model, the relevant microservices and base microservice model's (mandatory features') microservices are prepared for method calls;
- The optional microservice methods that are linked directly to an optional feature are associated to the relevant task(s) located in that feature's partial BPMN. The mandatory microservice methods that are linked directly to a mandatory feature are associated with the tasks located in the base BPMN. As a result, according to the changes made to the variability model, the microservice model methods are also prepared to be called by the BPMN.

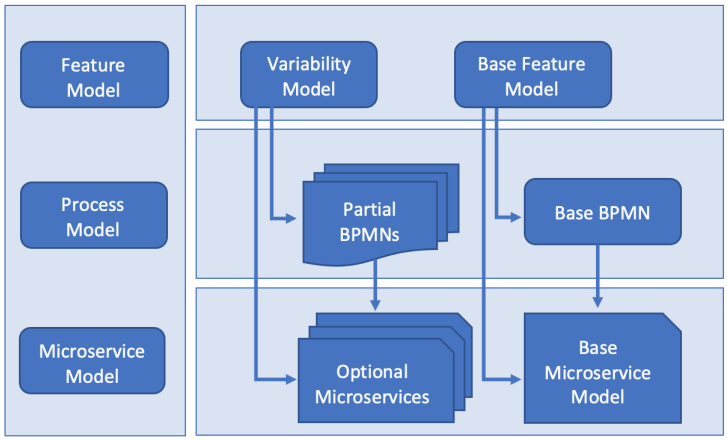


**Figure 3.** Models used in MSDeveloper.

MSDeveloper is shaped to carry the following properties:

- The environment is flexible and supports feature-based development;
- Variability resolution and related constraints propagate through all the models;
- BPMN is automatically configured as a result of variability resolution;
- Performing the microservice method through the BPMN provides the final steps in executability.

Product engineering is the part that corresponds more to low-code development, especially if a mature domain exists. A methodological flow of activities can also be presented for this task. However, the flow of activities does not prescribe a strict order. A natural start from higher-level abstraction models, and the development activities concerning them, are natural. As the inclusive feedback arrows imply, modifying any model after any assessment is possible. MSDeveloper adapts the activity flow resembling the low-code methodology for the product development in Figure 4.

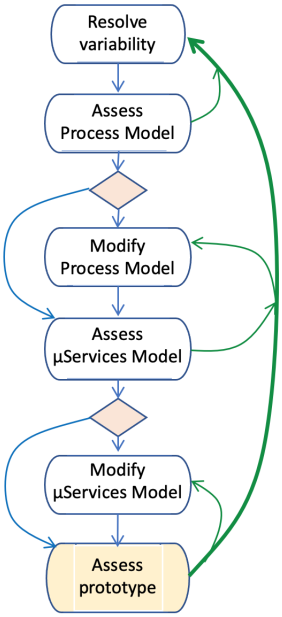


**Figure 4.** The low-code development methodology.

Consistency is an important criterion for both domain engineering and product engineering outcomes. As stated above, variability resolution and related constraints propagate through all the models, which supports the consistency of the output. After variability decisions, which are handled as a product engineering phase, the temporary outcome in

the process and the microservice models can be reviewed. Until the desired consistency is achieved, the developers can iterate through the activities of variability modifications and processes and microservice model inspections and modifications.

## 5. Experimental Study

In this section, we present our experimental study for evaluating the proposed methodology with student projects. Students used supportive prototype tools to model target systems by adhering to the methodology.

### 5.1. Development Environment

We set up an environment where Eclipse IDE for Java Developers (includes Incubating components) version December 2020 (4.18.0) is employed with the following plugins:

- FeatureIDE Plugin, Version: 3.7.0.202010141034;
- BPMN Plugin, Version: 1.5.2.SNAPSHOT-v20200602-1600-B1;
- Maven Plugin, Version: 1.17.1.20201207-1112

Moreover, we developed domain-independent and generic jar files that allow transitions (from feature model to BPMN and from BPMN to microservice method calls) and domain-related SpringBoot applications. Software structures for the selected domain and webinar system (feature model, auto-generated variability model, process model, and microservice model) are available in Ref. [29].

### 5.2. Selected Domain: Webinar System

Webinar is a combination of the words "web" and "seminar". It is used for systems that allow users to organize and participate in a video workshop, lecture, or presentation in an online environment. With the current COVID-19, such systems have become a part of our lives. Due to its popularity, we chose webinar systems for this experiment's domain.

Our chosen feature set for the webinar system contains the following features: live and pre-recorded video streaming, mobile accessibility, chat messaging, file sharing capabilities, virtual whiteboards, shared screens, virtual waiting rooms, breakout rooms, pass-presenter tools, polling tools, app integration, audio and video recording, in-app conference registration, automated reminders, Q and A tools, engagement analysis for attendance and lead generation, in-app offers, integration with live-streaming social media platforms, active speaker view, AI and gamification features, and backgrounds to create a branded virtual environment [30].

### 5.3. The Experiment: Webinar System Development

We conducted the experimental study in two phases. We asked the participants to complete the tasks given in Table 3 for Phase 1 and Phase 2.

**Table 3.** The experimental study phases.

| Phase 1 | Phase 2 |
| --- | --- |
| Domain Engineering Stages:<br>-Preparation of the feature model;<br>-Preparation of the variability model;<br>-Preparation of the complete BPMN;<br>-Preparation of the base BPMN;<br>-Preparation of the partial BPMNs. | Domain Engineering Stages:<br>-Preparation of the domain-specific methods;<br>-Invocation of all the related methods over the base BPMN;<br>-Invocation of the related methods from the partial BPMNs;<br>Product development stages;<br>-Selection of the variability model and automatic development of the product;<br>-Execution of the auto-generated product-specific BPMN model;<br>-Observation of correctness in the execution of related methods. |

#### 5.3.1. Participants

We conducted the experiment at the Department of Computer Engineering of Middle East Technical University during the 2020–2021 spring semester; a graduate class (CENG551

System Development with Abstract Design) and a senior class (CENG454 Introduction to Software Architecture) participated in this experiment. We asked students to form groups of two or three students. We gave groups a document [31] that included a system specification for the webinar system's design, along with the user manual for the development platform to be used in this experiment. The manual included details about the installed plugins in the provided development environment and general instructions about how to operate the environment.

After the completion of the project, we asked participants to answer some questions about their experience, provided development environment, and the system they designed. We conducted this questionnaire through the Internet. A total of 30 students participated in the questionnaire. The survey questions are given in Appendix A and the answers are presented in Ref. [32].

#### 5.3.2. Phase 1

We introduced the development setting to the developers before sharing the experimental webinar system domain. The shared feature model, which takes place in the shared workspace, is depicted in Figure 5, the shared variability model (which automatically performs all the controls regarding the features and constraints) is represented in Figure 6, and the shared BPMN model is shown in Figure 7.

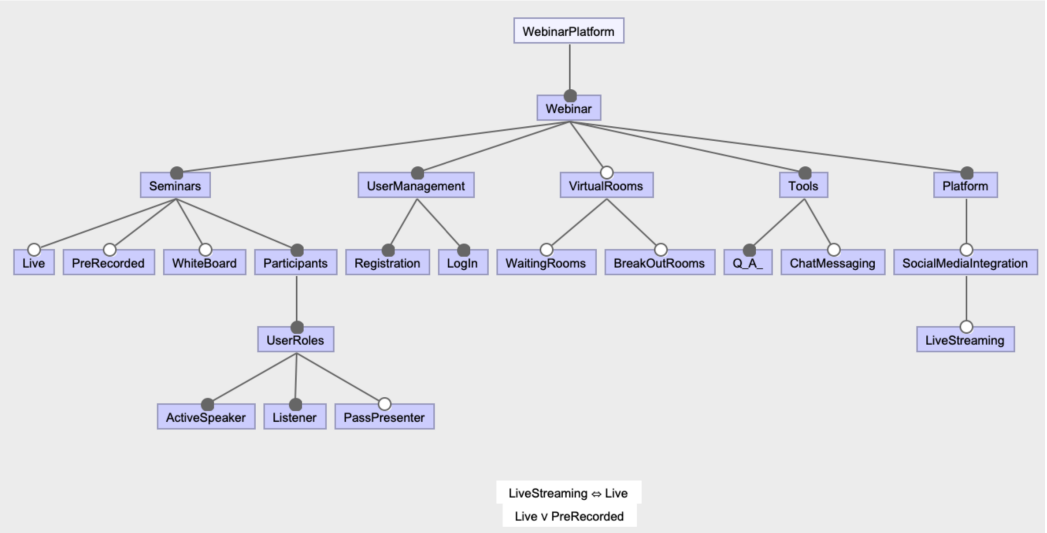


**Figure 5.** Webinar system domain feature model.

We asked developers to enrich the webinar system domain with optional and mandatory features to develop it towards a mature domain. The participants added the features they deemed necessary and enhanced the feature model by defining the constraints among the features. In this stage, they were able to perform visual modeling by using the drag-and-drop capability of the framework. Then, they prepared the variability model through support from the facility provided by the framework. A representation of the variability model is shown in Figure 6. After that, they prepared the base and the partial BPMNs for each optional feature.

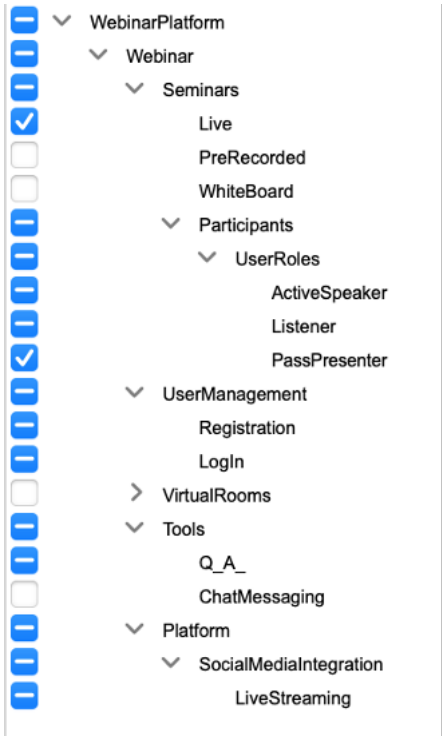


**Figure 6.** Variability model of webinar system domain.

After these preparatory stages, when the developers generated the product, they were able to observe that the resultant BPMN diagram was drawn according to the selections made in the variability model. The selected features were linked with corresponding process assets in the BPMN model. Additionally, we excluded assets that were directly linked with the unselected features from the BPMN Model.

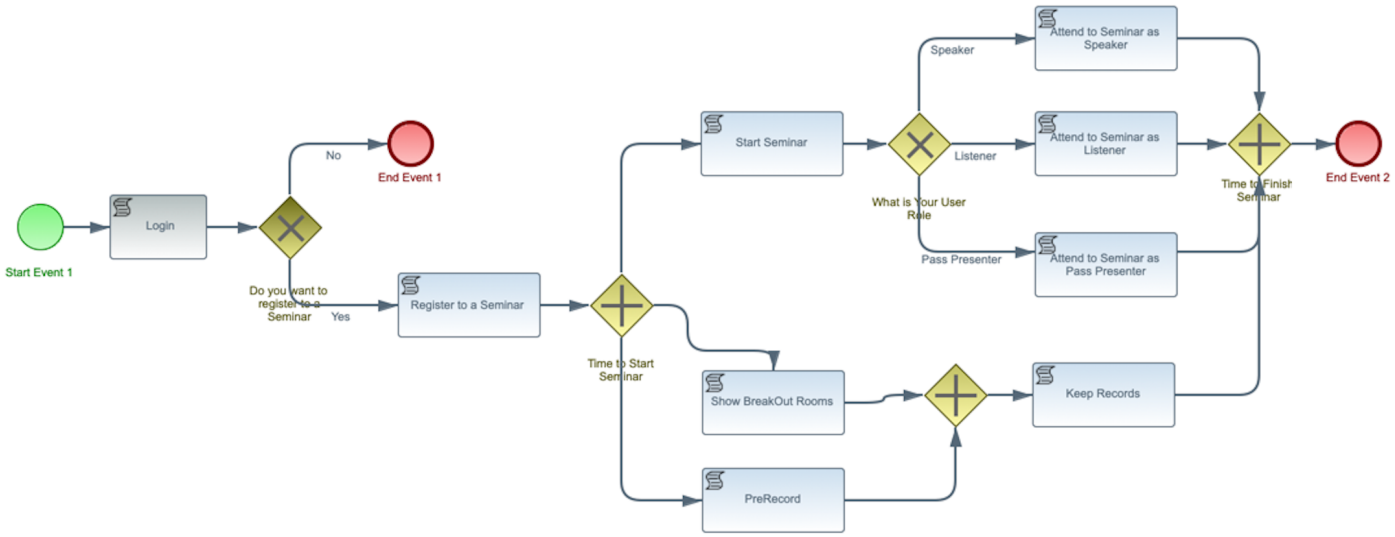


**Figure 7.** BPMN of webinar system domain.

### 5.3.3. Phase 2

We asked developers to perform method calls over the scripts corresponding to the tasks included in the BPMN. We asked them to call these methods by writing Java methods within these scripts. After, we asked them to add method calls to the partial BPMNs. As a result, the choices to be made on the variability model have a direct effect on the BPMN

and the methods called over the BPMN. In this phase participants used methods from Java classes that can directly represent a set of microservices.

In the final stage, we asked the developers to generate products with different configurations on the development environment they prepared for the webinar system. In this stage, they experienced the ability to produce products that differ according to the choices they made simply by updating the variability model.

## 6. Results

### *6.1. Experimental Study Results*

After the completion of the project, the participants were asked to fill out the survey questions. We analyzed the information we gathered from the participants via questionnaires. The results of the surveys are analyzed below. It should be noted that these participants are not professionals. This experiment group with no familiarity with low-code development rated their experience on a scale of 1 to 10, 1 being the hardest and 10 being the easiest, and they were asked to answer questions.

We can deduce from the following results stated in Figure 8 that half of the participants who have never developed in a low-code development environment do not find this environment very easy. On the other hand, it is seen that 64% of the participants have a positive opinion about the framework that we have prepared. Moreover, 86% of respondents think that they understood the framework well. This is encouraging feedback that suggested that we were able to explain our methodology and tools to the participants well. We see that after the domain-oriented development processes, 79% of them find it easy to generate the products through different configurations corresponding to low-code development.

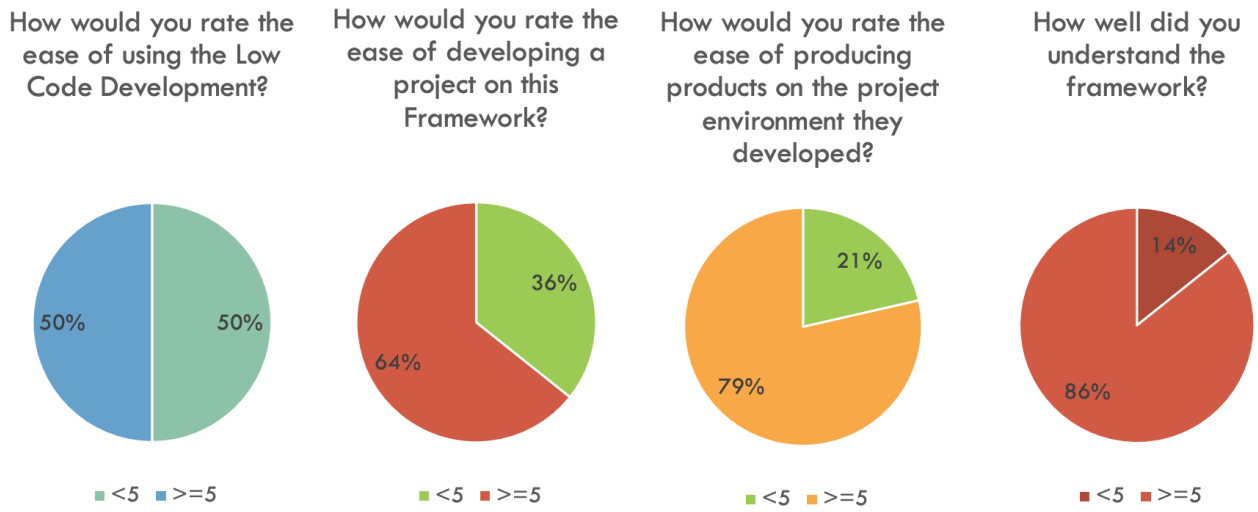


**Figure 8.** Easiness of the framework.

When we asked the participants about their experiences regarding the transitions among models, we encountered the results in Figure 9. The answers we obtained are not very surprising. Transitions between the feature model and BPMN model cover the creation of the complete BPMN and the preparation of the base and partial BPMNs. In summary, it includes drawing a business process model covering the entire domain and more. For this reason, only 57% of people think that this transition is easy. On the other hand, operations involving method calls from the BPMN model were found easy by 86%. We think that this is due to the fact that method calls from within the tasks defined in the BPMN model can be made through the easy-to-use GUI of the framework.

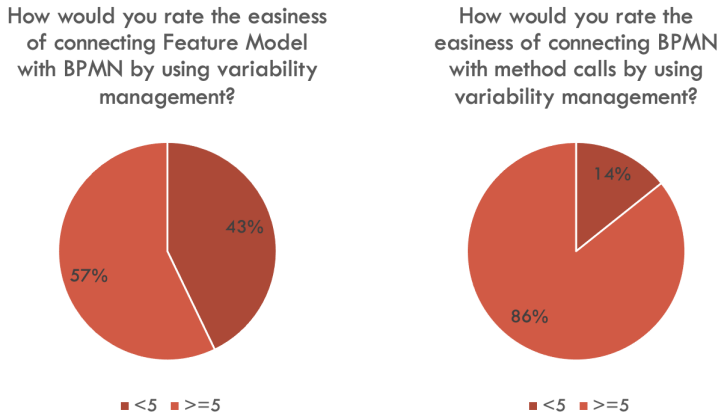


**Figure 9.** Questions about transitions between models and the responses .

One of the things we were most curious about during the experiment was the time spent. The answers we received to the questions we asked for this purpose are given in Figure 10. A total of 62% of respondents said they spent less than or equal to 10 h developing their webinar system domain. When generating the products with different configurations, 86% explained that they spent less than or equal to 6 h. We inferred that the time spent here is due to an immature domain because the framework completes the creation process of products developed in different configurations in milliseconds. However, the participants, who probably did not have a chance to develop a mature domain, encountered errors, corrections, etc. So, this step may have taken a long time.

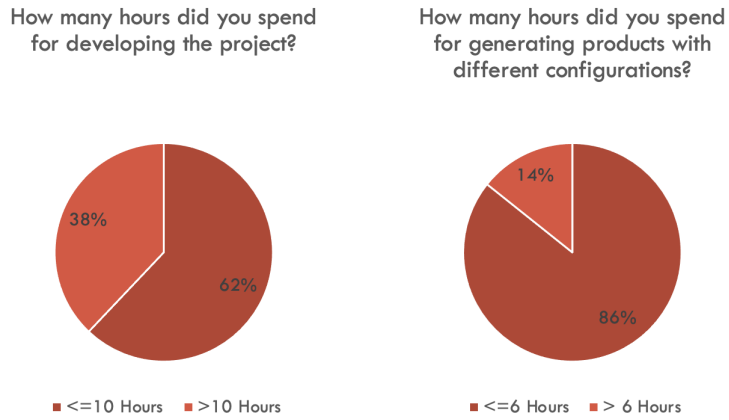


**Figure 10.** Questions about required time and the responses.

### *6.2. Professionals' Remarks*

The methodology and the framework were also shared with the professionals. In a video, the framework was introduced, and the processes of adding an optional feature on a mature domain and the effects of this feature addition on the feature, variability, and the BPMN models, as well as the method calls, are shown [33].

After watching the video, the professionals were asked to answer some questions about their opinions. This survey was conducted using the Internet. A total of 32 professionals participated in the survey. The survey questions are given in Appendix B and the answers are presented in [34]. The profile of the professionals (their work experience, company profile, and current working type) is shown in Figure 11.

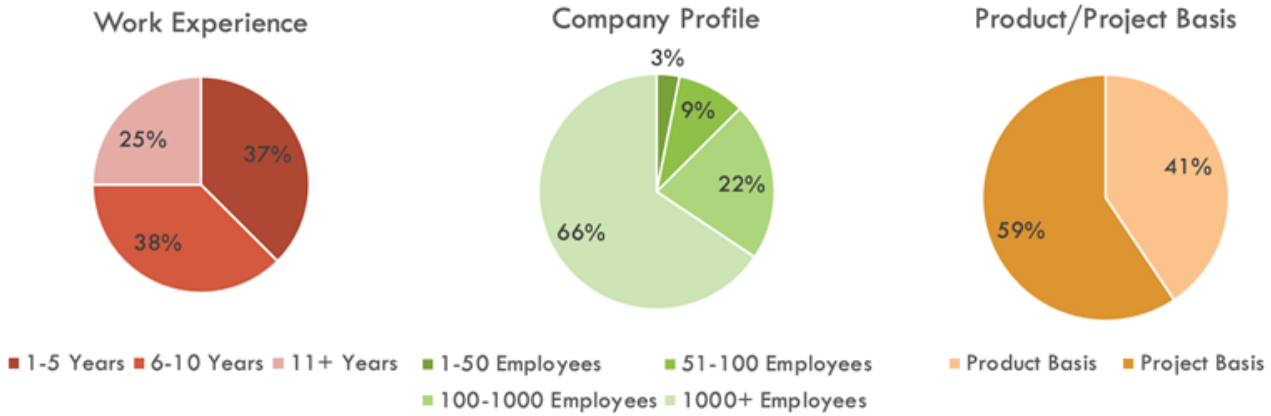


**Figure 11.** Profile of the professionals.

In Figure 12, we see the answers given by the product-based working professionals regarding questions related with the products they are working on. When we ask the question "Are you maintaining a single version (the most recent version) of the product you are working on?", 69% of them answered "No, Multiple Versions". That means most of them maintain the products with different configurations. Additionally, 69% of them find it difficult to see the overall picture and 92% of them have difficulties because of the size of the product, especially when they want to add a new feature to the product. After watching the video that we introduced in our methodology on the framework, 100% of them answered "Yes" to the question "Specific to the field you are working in, would a Framework be useful?".

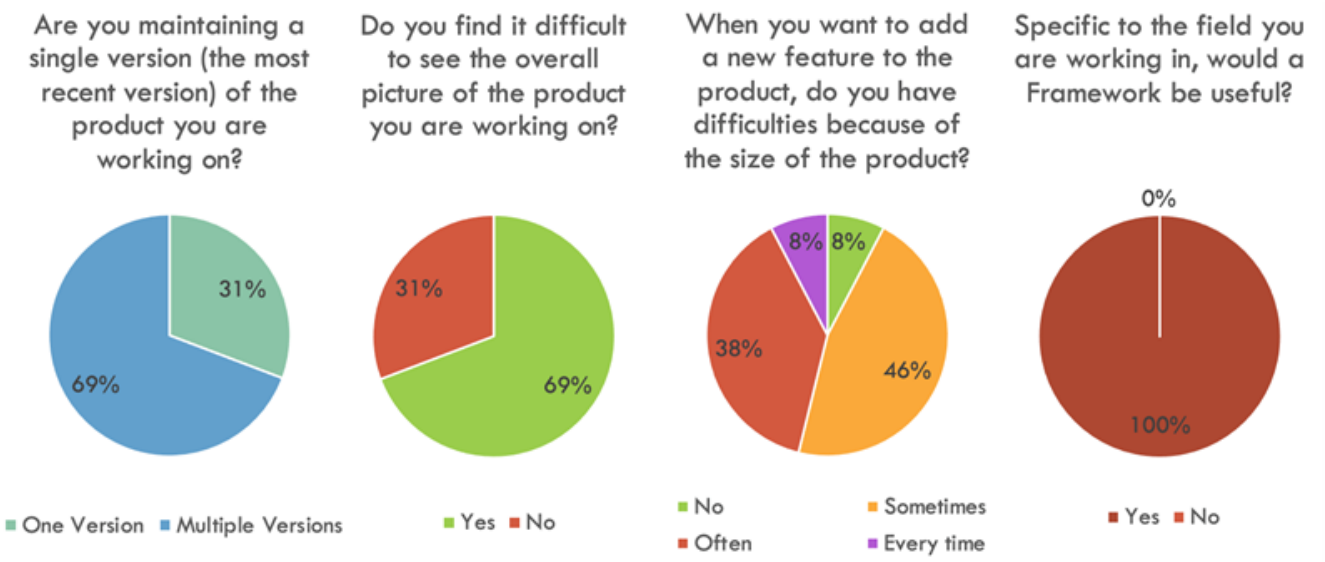


**Figure 12.** Product-based working professionals' answers.

In Figure 13, we see the answers given by all the professionals about the easiness of the framework in different perspectives, including developing a domain, creating a product, adding a feature, connecting a feature model with BPMN, and connecting BPMN with method calls. The evaluations of professionals with an average of 7.73 years of work experience are around 8 points, and these responses are greater than students' evaluations.

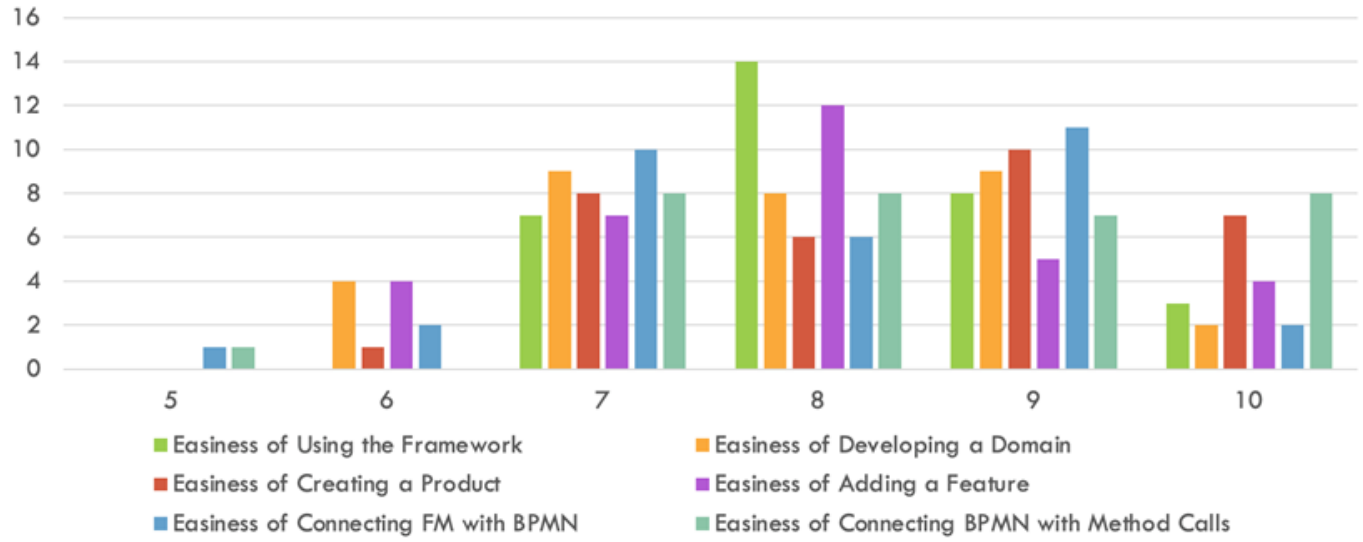


**Figure 13.** Easiness of the framework.

An open-ended question is also asked: "What are your views on the suitability of this methodology for professional life?". Some answers are listed below:

- In general, it will be useful for the structural standardization of the code. Models created by experienced teams will automatically ensure that less experienced employees stay within the quality constraints;
- It can be used easily and effectively in many large and small projects, and the time it saves in the development process cannot be underestimated;
- Low-code libraries make software development processes much easier and shorter today. Low-code development will become much more widespread in the future of the rapidly developing software industry;
- In general, the requirements are analyzed, a software/system is designed on top of it, and the whole development process continues, both waterfall and iterative. The Variability-Guided Development method seems to be able to parallelize both high- and low-level designs in waterfall or iterative processes. At the same time, it is a method that will minimize the workload when a new project is desired with the capabilities already in the product pool. It has an important place in creating backlogs for new features that need to be implemented. When Software Product Line is considered, maybe it will be helpful to deliver a project without the need for a developer;
- It can minimize the deficiencies that may occur during the manual management of multiple versions and the varying configurations of these versions. In addition, it will reduce the time spent managing the configurations with each changing version;
- It is very logical and solution-oriented because the project I am working on needs a framework just like this one. If given the opportunity and time, I would consider using it;
- It will be very useful, and it will save a lot of time since there are very similar projects with different configurations and code bases.

## 7. Discussion and Conclusions

The experiments and expert opinions support the usability of our proposal. The claim to offer a generic capability for the development of professional software would be validated through further experience, especially in industrial media. Expected experience is the definition of various domains and the creation of products in commercial settings. Nevertheless the conducted study provides a proof of concept. Our future work will continue in this direction to gain wider acceptance.

It was stated in the previous section that 100% of the participating professionals would find such a framework useful. Considering the difficulties of creating a mature domain in the professional world, it will be necessary to facilitate the domain development process. We aim to find new fields of study for the suitability of the domain development process with agile approaches, for example feature-driven development.

All features and constraints depend on the feature model, and all other layers are developed in accordance with corresponding constraints. This makes the whole process easier. For example, the function of a microservice, and whether it will be mandatory or optional, are very clearly revealed before starting its development. This situation is similar for the BPMN-model layer.

The framework we developed in academic settings is still in its infancy. However, by understanding the literature, examining the previously developed tools in detail, and by correctly understanding the needs of users, we aim to prepare a tool that is more useful and beneficial. According to the survey results, even though it is a new framework, the framework was found to be useful and promising. In the upcoming period, we will continue our work by evaluating the feedback we received from the participants and the remarks of the professionals.

Within the scope of this study, the users who mostly participated in the experiment with the role of domain developer were asked to prepare partial BPMNs for optional features during the transition from the feature model to the process model. Considering that all the other stages were conducted through graphical user interfaces, we can observe that this is the only part in which the participants were challenged; therefore, there was a need for an improvement on this matter.

We can conclude that the framework was successful with its low-code development goal. Microservices were appropriate to support the lower-level layer of the proposed architecture because of their granularity. We demonstrated the use of the environment for the product engineering task. However, to complete the spectrum for general-purpose development, domain engineering tasks also need to be experimented on. Our experimentation provided a prepared domain; however, the simple mechanisms adopted, while shown to be suitable, also indicate that effort would be involved in their development—domain engineering. Nevertheless, future work is required to experiment with this phase.

It is not difficult to realize that the environment is suitable for general purpose development. We are planning to continue our work in two avenues: integrating the tools to achieve a stand-alone and consistent development environment and conduct experimentations for the domain engineering phase. For this, the project groups will identify different domains and populate them with feature, process, and microservice models.

**Author Contributions:** Conceptualization, B.K.D., A.C., M.C.K. and A.H.D.; data curation, B.K.D.; formal analysis, B.K.D.; funding acquisition, B.K.D.; investigation, B.K.D., A.C., M.C.K. and A.H.D.; methodology, B.K.D., A.C., M.C.K., S.N. and A.H.D.; project administration, B.K.D.; resources, B.K.D.; software, B.K.D.; supervision, A.H.D.; validation, B.K.D., A.C. and A.H.D.; visualization, B.K.D., A.C. and A.H.D.; writing—original draft, B.K.D., A.C., M.C.K., S.N. and A.H.D.; writing—review and editing, B.K.D., A.C., M.C.K., S.N. and A.H.D. All authors have read and agreed to the published version of the manuscript.

**Funding:** The APC was funded by ASELSAN.

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** "The development workspace" at https://github.com/betulkuruoglu/developmentWorkspace (accessed on 7 October 2022).

**Conflicts of Interest:** Betul Kuruoglu Dolu is employed by ASELSAN.

## Abbreviations

The following abbreviations are used in this manuscript:

| | |
|---|---|
| BPMN | Business Process Modeling Notation |
| CRM | Customer Relationship Management |
| GUI | Graphical User Interface |
| IDE | Integrated Development Environment |
| MM | Microservice Model |
| OVM | Orthogonal Variability Model |
| SoA | Service-Oriented Architecture |
| SPL | Software Product Line |
| SPLE | Software Product Line Engineering |
| VM | Variability Model |

## Appendix A

Framework: Low-code development environment shared with you

Project: Interconnected feature model–configuration–BPMN–method calls

Product: Each "Configuration—BPMN—Method Calls" after you run your projects by changing the configuration

1. Complete following fields:
   - Course code;
   - Group number.
2. Do you have any previous experience on a low-code development environment? (Answer this question per group member);
3. On a scale of 1 to 10, 1 being the hardest and 10 being the easiest, how would you rate the easiness of using the framework shared with you?
4. On a scale of 1 to 10, 1 being the hardest and 10 being the easiest, how would you rate the easiness of developing projects using the framework?
5. On a scale of 1 to 10, 1 being the hardest and 10 being the easiest, how would you rate the easiness of creating a product by changing the feature model configuration?
6. On a scale of 1 to 10, 1 being the lowest and 10 being the highest, how well did you understand the framework?
7. Plugins that offer you a user interface to prepare the feature model and BPMN with constraints are given in the framework. On a scale of 1 to 10, 1 being the hardest and 10 being the easiest, how would you rate the easiness of creating a feature model and BPMN by using the Framework?
8. On a scale of 1 to 10, 1 being the hardest and 10 being the easiest, how would you rate the easiness of switching from feature model to BPMN by using variability management?
9. On a scale of 1 to 10, 1 being the hardest and 10 being the easiest, how would you rate the easiness of switching from BPMN to method calls by using variability management?
10. If you would have to develop similar products by changing the configuration, would you consider developing them by using a low-code development environment?
11. Specify how many hours you spent:
    - For learning the framework;
    - For developing the project;
    - For creating the product according to a specific configuration.
12. Which features do you think are essential for a webinar system?
13. What do you think about the fidelity of your project? (Fidelity searches for the answer to the following question: Does the project represent a real webinar system?)

## Appendix B

1. What is your current position?
2. How many years of work experience do you have in the information technologies and software development industry?
3. In which field(s) did you receive your university degree(s)?
4. What is the size of your firm (number of employees)?
5. Are you working on a project or product basis?
6. Have you taken part in more than one project in the company you work for?
7. What is the number of projects developed by your company in the field you are working in?
8. If the number of projects developed by your company in the field you work in is two or more, what is the percentage of the similarities (area, requirements, coding, testing, etc.) of these projects?
9. Are you maintaining a single version (the most recent version) of the product you are working on?
10. Do you find it difficult to see the overall picture of the product you are working on?
11. When you want to add a new feature to the product you are working on, do you have difficulties due to the size of the product?
12. Would it be useful to have a framework that keeps assets specific to the field you are working in, such as common requirements, business processes, and code bases, and allows these assets to be prepared with a common infrastructure?
13. On a scale of 1 to 10, 1 being the hardest and 10 being the easiest, can you evaluate the easiness of the framework shared with you?
14. On a scale of 1 to 10, 1 being the hardest and 10 being the easiest, can you evaluate the easiness of developing a domain using the framework shared with you?
15. On a scale of 1 to 10, 1 being the hardest and 10 being the easiest, can you rate the easiness of creating a product using the framework shared with you?
16. On a scale of 1 to 10, 1 being the hardest and 10 being the easiest, how would you rate the easiness of creating a feature on this framework?
17. On a scale of 1 to 10, 1 being the hardest and 10 being the easiest, can you evaluate the easiness of transitioning from a feature model to BPMN using variability management?
18. On a scale of 1 to 10, 1 being the hardest and 10 being the easiest, can you rate the easiness of transitioning from BPMN to method calls?
19. If you need to develop similar products by changing/adding features, would you consider developing using a low-code development environment?

## References

1. Waszkowski, R. Low-code platform for automating business processes in manufacturing. *IFAC-PapersOnLine* **2019**, *52*, 376–381. [CrossRef]
2. Sahay, A.; Indamutsa, A.; Di Ruscio, D.; Pierantonio, A. Supporting the understanding and comparison of low-code development platforms. In Proceedings of the 2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Portoroz, Slovenia, 26–28 August 2020; pp. 171–178.
3. Togay, C.; Dogru, A.H.; Tanik, J.U. Systematic component-oriented development with axiomatic design. *J. Syst. Softw.* **2008**, *81*, 1803–1815. [CrossRef]
4. Creatio. Creatio Studio Platform Overview. 2022. Available online: https://www.creatio.com/studio (accessed on 25 August 2022).
5. Pega. Pega Platform Overview. Available online: https://www.pega.com/products/platform (accessed on 25 August 2022).
6. Salesforce. Salesforce Platform Overview. 2022. Available online: https://www.salesforce.com/eu/products/platform/overview/ (accessed on 19 August 2022).
7. Zoho Creator. Zoho Creator Platform Overview. 2022. Available online: https://www.zoho.com/creator/product-overview.html?src=hdd (accessed on 19 August 2022).
8. Vincent, P.; Iijima, K.; Driver, M.; Wong, J.; Natis, Y. *Magic Quadrant for Enterprise Low-Code Application Platforms*; Gartner Report; Gartner, Inc.: Stamford, CT, USA , 2019.

9. Ihirwe, F.; Di Ruscio, D.; Mazzini, S.; Pierini, P.; Pierantonio, A. Low-code engineering for internet of things: A state of research. In Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings, Virtual, 16–23 October 2020; pp. 1–8.
10. Kaya, M.C.; Cetinkaya, A.; Dogru, A.H. Off-the-shelf connectors for interdisciplinary components. *J. Integr. Des. Process. Sci.* **2018**, *22*, 35–53. [CrossRef]
11. Kaya, M.C.; Suloglu, S.; Tokdemir, G.; Tekinerdogan, B.; Dogru, A.H. Variability incorporated simultaneous decomposition of models under structural and procedural views. In *Software Engineering for Variability Intensive Systems*; Auerbach Publications: Boca Raton, FL, USA, 2019; pp. 95–115.
12. Suloglu, S.; Kaya, M.C.; Cetinkaya, A.; Karamanlioglu A.; Dogru, A.H. Cloud-Enabled Domain-Based Software Development. In *Software Engineering in the Era of Cloud Computing*; Springer: New York, NY, USA, 2020; pp. 109–130.
13. Böhm, C.; Jacopini, G. Flow diagrams, Turing machines and languages with only two formation rules. *Commun. ACM* **1966**, *9*, 366–371. [CrossRef]
14. Lewis, J.; Fowler, M. Microservices, a Definition of This New Architectural Term. 2014. Available online: https://martinfowler.com/articles/microservices.html (accessed on 19 August 2022).
15. Kyle, B. Beyond Buzzwords: A Brief History of Microservices Patterns. 2016. Available online: https://developer.ibm.com/articles/cl-evolution-microservices-patterns/ (accessed on 19 August 2022).
16. Dogru, A.H.; Tanik, M.M. A process model for component-oriented software engineering. *IEEE Softw.* **2003**, *20*, 34–41. [CrossRef]
17. Pohl, K.; Böckle, G.; Linden, F. Software Product Line Engineering: Foundations, Principles, and Techniques Springer. 2005. Available online: https://link.springer.com/book/10.1007/3-540-28901-1 (accessed on 19 August 2022).
18. Van Gurp, J.; Bosch, J.; Svahnberg, M. On the notion of variability in software product lines. In Proceedings of the Working IEEE/IFIP Conference on Software Architecture, Amsterdam, The Netherlands, 28–31 August 2001; pp. 45–54.
19. Sinnema, M.; Deelstra, S.; Nijhuis, J.; Bosch, J. Covamof: A framework for modeling variability in software product families. In *Proceedings of the International Conference on Software Product Lines*; Springer: Berlin/Heidelberg, Germany, 2004; pp. 197–213.
20. Cetinkaya, A.; Suloglu, S.; Cagri Kaya, M.; Karamanlioglu, A.; Tokdemir, G.; Dogru, A.H. An Experimental Study on Decomposition: Process First or Structure First? In *Proceedings of the International Symposium on Business Modeling and Software Design*; Springer: Lisbon, Portugal, 2019; pp. 279–289.
21. Richardson, C.; Rymer, J.R. *Vendor Landscape: The Fractured, Fertile Terrain of Low-Code Application Platforms*; FORRESTER: Cambridge, MA, USA, January 2016.
22. Basciani, F.; Iovino, L.; Pierantonio, A. MDEForge: An extensible web-based modeling platform. In Proceedings of the 2nd International Workshop on Model-Driven Engineering on and for the Cloud, CloudMDE 2014, Co-located with the 17th International Conference on Model Driven Engineering Languages and Systems, MoDELS 2014, CEUR-WS, Valencia, Spain, 30 September 2014; Volume 1242, pp. 66–75.
23. Appian. Appian Platform Overview. 2022. Available online: https://appian.com/platform/overview.html (accessed on 19 August 2022).
24. Kissflow. Kissflow Platform Overview. 2022. Available online: https://kissflow.com/platform/ (accessed on 19 August 2022).
25. Mendix. Mendix Platform Overview. 2022. Available online: https://www.mendix.com/low-code-guide/ (accessed on 19 August 2022).
26. Microsoft Power Apps. Microsoft Power Apps Platform Overview. 2022. Available online: https://powerapps.microsoft.com/en-us/ (accessed on 19 August 2022).
27. Outsystems. Outsystems Platform Overview. 2022. Available online: https://www.outsystems.com/ (accessed on 19 August 2022).
28. Suh, N.P. *Axiomatic Design: Advances and Applications*; Oxford University Press: New York, NY, USA, 2001.
29. Dolu, B.K. The Development Workspace. 2022. Available online: https://github.com/betulkuruoglu/developmentWorkspace (accessed on 7 October 2022).
30. Webb, T. Best Virtual Conference Platforms for Online Events. 2022. Available online: https://getvoip.com/blog/virtual-conference-platforms/ (accessed on 19 August 2022).
31. Cetinkaya, A. Project Specification and Development Platform Manual. 2020. Available online: https://drive.google.com/file/d/1sJC27hAGwfBIMpAuV4-cUeGqzbyK0hVF/ (accessed on 19 August 2022).
32. Cetinkaya, A. Low-Code Development Environment Survey/Responses. 2020. Available online: https://docs.google.com/spreadsheets/d/1itr5W4dWq3lUBHEYmQ2okDDUR8m1iWs8oMFN-XcbvPg/ (accessed on 19 August 2022).
33. Dolu, B.K. Variability-Guided Low-Code Development Environment Introductory Video. 2021. Available online: https://vimeo.com/664041103/ea798f30bc (accessed on 19 August 2022).
34. Dolu, B.K. Low-Code Development Environment Survey/Responses. 2021. Available online: https://docs.google.com/spreadsheets/d/1BWFIp2CFhmcXRiXqyXQdJayz5IFqMWtM0SF4dUJzptI (accessed on 19 August 2022).