Business Intelligence Management Studies
Vol. 11, No.44, Summer 2023; P 225-257
ims.atu.ac.ir
DOI: 10.22054/ims.2023.70563.2245

ATU PRESS

Original Research
Accepted: 20-06-2023
Revised: 13-03-2023
Received: 17-10-2022
eISSN: 2821-0816
ISSN: 2821-0964

Mutual Roles of Model-Driven Software Engineering and Low-Code Development Platform

Leila Samimi Dehkordi
Assistant Professor, Department of Computer Engineering, Faculty of Engineering, Shahrekord University, Shahrekord, Iran.

Abbas Horri *
Assistant Professor, Department of Computer Engineering, Faculty of Engineering, Shahrekord University, Shahrekord, Iran.

Abstract

1. Introduction

Software engineering is an engineering system that aims to educate, research, and apply methods to develop applications for increasing software productivity and quality and reducing the cost and production time (Kung, 2013). One of the software engineering methods that has received attention in recent years is using Low-Code Development Platforms (LCDP) (Alamin et.al, 2023). LCDPs use a graphical user interface to develop software instead of traditional programming. These types of platforms are suitable tools for organizational companies to reduce development costs and time to market (Tisi et al., 2020). Increasing the level of abstraction in order to reduce the cost of development is exactly the same goal that "Model-Driven Software Engineering" (MDSE) pursues. MDSE is an approach in software engineering where models are used not only as documentation but also for automatic code generation (Brambilla et al., 2017). The MDSE methodology has matured and its best practices can be used for the new field of LCDP (Verbruggen and Snoeck, 2023). At the same time, LCDP has been of great interest in the last few years, and migrating from Model-Driven Engineering to cloud spaces to create LCDPs can be applicable and appropriate (Ruscio et

* Corresponding Author: Samimi@sku.ac.ir
How to Cite: Samimi-Dehkordi, L., Horri, A. (2023). Mutual Roles of Model-Driven Software Engineering and Low-Code Development Platform, *Journal of Business Intelligence Management Studies*, 11(44), 225-257.

al., 2022).

Research Question(s)

1. What features can be considered for common LCDPs?
2. What is the position of LCDPs compared to MDSE?
3. What are the prerequisites for migrating modeling languages from MDSE to LCDP?

2. Literature Review

2.1. Theoretical foundations of research

Model-Driven Engineering and a Low-Code Development Platform have both been introduced with the goal of rapid product delivery with minimal programming (Ruscio et al., 2022). However, MDSE is more mature than LCDP (Verbruggen and Snoeck, 2023). Mendix, one of the pioneers in the LCDP field, has presented a LCDP manifesto including 9 principles, which are model-driven development, collaboration, agility, cloud computing, openness, multi-user development, experimentation, governance, and community (Kenneweg et al., 2021). To investigate the features of the LCDP platforms, six stages for developing an application has introduced, which are domain modeling, user interface definition, business logic specification, integration with external services, deployment, and maintenance (Sahay et al., 2020).

2.2. Related Work

2.2.1. Researches related to LCDP and MDSE

Cabot stated that the LCDP approach is the same as MDSE and it has been changed only with the aim of attracting the audience and better understanding the name of the approach (Cabot, 2020). Khorram et al. stated that LCDPs are based on MDSE, in which system design with visual modeling and automatic production of the final executable system has been introduced as a common feature of both approaches (Khorram et al., 2020). In the Locomote framework research, it is stated that developers can take advantage of MDSE principles, but as scalability is one of the serious problems in MDSE, this challenge is more evident in LCDP (Tisi et al., 2020). Alamin et al. stated that LCDP is inspired by MDSE, and development is done using abstract representations instead of focusing on algorithmic calculations (Alamin et al., 2021). Ruscio et al., have classified five research areas,

in which the differences between MDSE and LCDP, including end users and application scope, are stated (Ruscio et al., 2022).

2.2.2. Well-known LCDPs

In Appsheet, a variety of tools and services, including data-driven applications, can be easily developed on top of the Google cloud database (Käss et al., 2023). In SwiftUI, it is possible to create a user interface for any Apple device by means of a declarative syntax, drag-and-drop support, and real-time preview (Nekras, 2022). Honeycode is a spreadsheet component and a set of templates for creating simple web-based applications (ElBatanony and Succi, 2021). PowerApps is a no-code platform for business users that starts with the data model and business processes and goes on to automatically generate responsive portable applications. (Gürcan and Taentzer, 2021). OutSystems is a low-code development platform that enables the development of desktop and mobile applications (Martins et al., 2020). Mendix is a low-code development platform where all features can be accessed via drag-and-drop functionality (Gürcan and Taentzer, 2021). KissFlow is a cloud-based workflow automation software platform that helps users build and modify automated enterprise applications (Hili and Oliveira, 2022). Appian is one of the oldest LCDPs that enables the creation of mobile and web applications through personalization tools, built-in team collaboration tools, task management, and social networking (Vincent et al., 2019).

3. Methodology

The present research method is practical in terms of purpose and descriptive survey in terms of nature. The purpose of this research is to investigate various approaches in the field of LCDPs and compare them with MDSE approaches. Based on this, three research questions were designed. The first question is to examine the characteristics of LCDPs. To answer the first question, two types of studies have been conducted to collect information about these platforms: (1) a review of articles from 2014 to 2023 and (2) a review of LCDP tools.

The second question is to examine the position of LCDPs compared to the MDSE approaches. The review of articles has been from 2014 onwards and the articles that have addressed both issues have been taken into consideration.

The main challenge for LCDPs is the commercial nature of these

platforms, which makes a limited community of users able to use them. MDSE tools are often academic and free. Consequently, the third question examined the requirements for moving from MDSE to LCDP. To answer this question, by introducing a case, the requirements of migration have been studied.

4. Conclusion

In this paper, the emerging approach of LCDP has been introduced. First, the important features of LCDPs have been reviewed and seven different tools were compared based on the reviewed features. Also, due to the common goal with the MDSE field, a comparison between these two fields has been presented and the position of LCDPs has been determined. Based on the MDSE benefits, the migration of modeling languages from the model-driven approach to the low-code development platform has been studied, and an example of migration has been investigated. One of the important limitations of this research is the lack of sufficient resources in the modern LCDP field. Most of the common platforms are commercial and there are few free platforms currently. Consequently, we suggested using the experiences of the model-driven field in the development of these types of platforms. For future work, it is necessary to study the characteristics and complexity of applications built using LCDPs, with the aim of evaluating the performance status of these platforms and reasoning about criteria such as their scalability and efficiency.

Acknowledgments

This article is derived from the results of the research project implemented under the contract number 5497/141 from the funds of Shahrekord University Research and Technology Vice-Chancellor.

Keywords: Model-Driven Engineering, Low-Code Development Platform, Cloud Computing.

مقاله پژوهشی تاریخ ارسال: ۱۴۰۱/۰۷/۲۵ تاریخ بازنگری: ۱۴۰۱/۱۲/۲۲ تاریخ پذیرش: ۱۴۰۲/۰۳/۳۰ ISSN: 2821-0964 eISSN: 2821-0816

مطالعات مدیریت کسب و کار هوشمند
سال یازدهم، شماره ٤٤، تابستان ١٤٠٢، ص ٢٢٥ تا ٢٥٧
ims.atu.ac.ir
DOI: 10.22054/ims.2023.70563.2245

نقش متقابل مهندسی نرم‌افزار مدل‌رانده و سکوی توسعه کم‌کد

لیلا صمیمی دهکردی استادیار گروه مهندسی کامپیوتر، دانشکده فنی و مهندسی، دانشگاه شهرکرد، شهرکرد، ایران.

عباس حری * استادیار گروه مهندسی کامپیوتر، دانشکده فنی و مهندسی، دانشگاه شهرکرد، شهرکرد، ایران.

چکیده

در چند سال اخیر، «سکوهای توسعه کم‌کد» در جلب‌توجه بازار رشد چشمگیری داشته‌اند. سکوهای کم‌کد، به توسعه برنامه‌های کاربردی به‌صورت بصری و روی ابر پرداخته و نیاز به کدنویسی دستی را کاهش می‌دهند. همچنین، توسط توسعه‌دهندگان غیرحرفه‌ای با دانش محدود در برنامه‌نویسی مورداستفاده قرار می‌گیرند و برای انجام این امر از مهندسی مدل‌رانده بهره می‌برند. در این مقاله، به سه پرسش با روش پیمایشی و بر اساس مستندات کتابخانه‌ای پاسخ داده شد. اول، ویژگی‌های سکوهای شناخته‌شده موردبررسی قرار گرفت تا مزایای این رویکرد بررسی شود. دوم، با توجه به وجوه مشترک سکوهای کم‌کد و رویکردهای مدل‌رانده، جایگاه سکوی کم‌کد در مقایسه با این رویکردها موردبررسی قرار گرفت و نقاط ضعف و قوت هر دو مشخص شود. سوم، به دلیل مزیت اصلی سکوی کم‌کد یعنی رایانش ابری و مزیت اصلی مدل‌رانده یعنی رایگان بودن ابزارها، به بررسی الزامات انتقال زبان‌های مدل‌سازی از مهندسی مدل‌رانده به سکوی توسعه کم‌کد پرداخته شد.

کلیدواژه‌ها: مهندسی نرم‌افزار مدل‌رانده، سکوهای توسعه کم‌کد، رایانش ابری.

مقاله حاضر مستخرج از نتایج طرح تحقیقاتی اجرا شده به شماره قرارداد ۵۴۹۷/۱۴۱ از محل اعتبارات معاونت پژوهش و فناوری دانشگاه شهرکرد می‌باشد.

* نویسنده مسئول : Samimi@sku.ac.ir

مقدمه

مهندسی نرم‌افزار، یک نظام‌مهندسی بوده که باهدف آموزش، پژوهش و به‌کارگیری روش‌ها جهت توسعه برنامه‌های کاربردی انجام می‌شود و به افزایش بهره‌وری و کیفیت نرم‌افزار و کاهش هزینه و زمان تولید می‌پردازد (Kung, 2013). یکی از روش‌های مهندسی نرم‌افزار که در سال‌های اخیر موردتوجه قرار گرفته است، استفاده از سکوهای توسعه کم‌کد[1] می‌باشد (Alamin et.al, 2023). این سکوها با افزایش تعداد فروشندگان ابری مانند گوگل، مایکروسافت و آمازون که راه‌حل‌هایی را برای توسعه و اجرای برنامه‌های نرم‌افزاری پیچیده با کدنویسی کم یا بدون کدنویسی ارائه می‌دهند، در حال افزایش هستند (Richardson and Rymer, 2014). سکوهای توسعه کم‌کد به‌جای برنامه‌نویسی سنتی، از واسط گرافیکی کاربر برای توسعه نرم‌افزارها استفاده می‌کنند. این نوع سکوها ابزارهای مناسبی برای شرکت‌های سازمانی جهت کاهش هزینه‌های توسعه محسوب می‌شوند (Tisi et al., 2020). کاهش هزینه عمدتاً به دو طریق امکان‌پذیر می‌شود: (۱) با ارائه‌ی نرم‌افزاری که توسعه‌دهندگان غیرحرفه‌ای (در برنامه‌نویسی) می‌توانند از آن استفاده کنند تا برنامه‌های کاربردی متناسب با نیاز خود را ایجاد کنند. (۲) با ارائه دادن سیستمی به برنامه‌نویسان جهت توسعه سریع کاربردهای کوچک برای این‌که زمان تحویل محصول به بازار کمینه شود (Richardson and Rymer, 2014).

اصطلاح «سکوی توسعه کم‌کد» توسط فارستر[2] در سال ۲۰۱۴ ابداع شد (Hey et al., 2014). سکوی کم‌کد، سکوی نرم‌افزاری است که تحویل سریع برنامه‌های کسب‌وکار را با حداقل برنامه‌نویسی دستی و حداقل سرمایه‌گذاری برای راه‌اندازی، آموزش و استقرار امکان‌پذیر می‌سازد (Richardson and Rymer, 2014). این اولین باری نیست که جامعه مهندسین نرم‌افزار تلاش دارد کدنویسی دستی را با ترکیب فناوری‌های توسعه و تولید خودکار کد کاهش دهد (Cabot, 2020). این هدف، دقیقاً همان چیزی است که مهندسی

1. Low-Code Platform Development (LCPD)
2. Forrester

نرم‌افزار از ابتدا به‌دنبال آن بوده است؛ همان‌طورکه گردی بوچ[1] گفته است، کل سابقه مهندسی نرم‌افزار راجع به افزایش سطح انتزاع است (Booch, 2018). افزایش سطح انتزاع به‌منظور کاهش هزینه توسعه تا استقرار برنامه‌های کاربردی دقیقاً همان هدفی است که «مهندسی نرم‌افزار مدل‌رانده[2]» هم دنبال می‌کند. مهندسی مدل‌رانده، نگرشی در مهندسی نرم‌افزار است که در آن از مدل‌ها نه‌تنها به‌عنوان مستندات بلکه برای تولید خودکار کد نیز استفاده می‌شود (Brambilla et al., 2017).

تولوانن و کلی[3] (۲۰۰۸) شیوه‌های توسعه نرم‌افزار را با توجه به نقش مدل‌سازی به شش دسته تقسیم می‌کنند: (۱) عدم استفاده از مدل‌سازی، (۲) مدل‌سازی صرفاً برای مستندسازی و تحلیل، (۳) مهندسی معکوس[4]، (۴) مهندسی رفت‌وبرگشت[5]، (۵) مدل‌سازی خاص دامنه[6] و (۶) اجرای مدل[7] (Tolvanen and Kelly, 2008). در روش‌های اول و دوم مدل‌سازی به‌طور مستقیم در تولید نرم‌افزار نقش ندارد. در مهندسی معکوس، برای تحلیل سریع‌تر نرم‌افزار، مدل‌ها از روی کد استخراج می‌شوند (Bruneliere et al., 2014). مهندسی رفت‌وبرگشت روش پرکاربرد دیگری در مهندسی مدل‌رانده است که تغییرات مدل و کد را با یکدیگر سازگار می‌کند (Hettel et al., 2008). رایج‌ترین روش مدل‌سازی خاص دامنه است که در آن، کد برنامه از روی مدل با استفاده از تولیدکننده‌های کد ایجاد می‌شود (Tolvanen and Kelly, 2008). در اجرای مدل، به‌جای تولید کد از مدل، خود مدل اجرا می‌شود و برای این منظور باید از ابزارهای مفسر مدل[8] استفاده کرد (Mayerhofer et al., 2013). به‌دلیل وجه اشتراک در هدف کاهش کدنویسی دستی بین مهندسی مدل‌رانده و توسعه کم‌کد، روش پنجم و ششم می‌توانند در

1. Grady Booch
2. Model-Driven Software Engineering
3. Tolvanen, J. P., Kelly, S.
4. Reverse Engineering
5. Round-trip Engineering
6. Domain-Specific Modeling
7. Model Execution
8. Model Interpreter

سکوهای توسعه کم‌کد نیز استفاده شوند.

می‌توان گفت سکوی توسعه کم‌کد از مهندسی مدل‌رانده الهام گرفته‌شده، همان‌طورکه مهندسی نرم‌افزار مدل‌رانده از ابزارهای مهندسی نرم‌افزار به کمک کامپیوتر[1] الهام گرفته شده است (Hey et al., 2014). با توجه به این‌که مهندسی مدل‌رانده به بلوغ رسیده و اصول علمی و فنی برای آن وجود دارد، می‌توان از تجربه‌های موفق آن برای حوزه جدید سکوی کم‌کد استفاده کرد (Verbruggen and Snoeck, 2023). درعین‌حال، سکوی توسعه کم‌کد در چند سال اخیر بسیار موردتوجه بوده و مهاجرت از مهندسی مدل‌رانده به سمت فضاهای ابری جهت ایجاد سکوهای کم‌کد می‌تواند مفید و مناسب باشد (Ruscio et al., 2022).

پرسش‌های اصلی پژوهش در زیر مطرح شده‌اند:

۱. چه ویژگی‌هایی را می‌توان برای سکوهای توسعه کم‌کد رایج در نظر گرفت؟

۲. جایگاه سکوهای توسعه کم در مقایسه با مهندسی نرم‌افزار مدل‌رانده چگونه است؟

۳. ملزومات انتقال زبان‌های مدل‌سازی از مهندسی مدل‌رانده به سکوی توسعه کم‌کد چیست؟

مبانی نظری پژوهش

مهندسی مدل‌رانده و سکوی توسعه کم‌کد هر دو باهدف تحویل سریع محصول با حداقل برنامه‌نویسی معرفی شده‌اند (Ruscio et al., 2022). در این میان، مهندسی مدل‌رانده به نسبت سکوی کم‌کد بلوغ بیشتری دارد و همین موضوع باعث شده است که توسعه‌دهندگان سکوی کم‌کد از تجربیات حوزه مدل‌رانده بهره ببرند (Verbruggen and Snoeck, 2023). از مفاهیم مهمی که در مهندسی مدل‌رانده مطرح هستند و در این پژوهش اهمیت دارند می‌توان به تولید کد، زبان خاص دامنه و خودکارسازی اشاره کرد.

1. Computer-Aided Software Engineering

تولید کد از ابزارهایی برای ایجاد کد از روی مدل‌ها، قالب‌ها و رابط کاربری استفاده می‌کند (Brambilla et al., 2017). زبان خاص دامنه، زبانی است که برای یک قلمرو، زمینه یا شرکت جهت ساده کردن وظایف، طراحی می‌شود (Mernik et al., 2005). در مهندسی مدل‌رانده، خودکارسازی یک اصل مهم است که از مدل بتوان برنامه کاربردی را ایجاد کرد یا به‌اجرا درآورد (Cabot, 2020). ازجمله سکوهای کم‌کد، می‌توان به «سکوهای بدون کد» اشاره کرد که به‌واسطه‌ی آن، توسعه برنامه‌ها از طریق پر کردن فرم‌ها یا درج اطلاعات در داخل یک ساختار از پیش تعریف‌شده انجام می‌شود (Rokis and Kirikova, 2022). سکوهای بدون کد انعطاف‌پذیری را محدود می‌کنند، اما سطح مشخصی از کیفیت و تجربه استاندارد و آشنا را برای همه کاربران تضمین می‌کنند (Cabot, 2020).

توسعه‌دهندگان حوزه جدید سکوی توسعه کم‌کد همان افراد جامعه مهندسی مدل‌رانده هستند که با معرفی این زمینه جدید، فناوری نرم‌افزار را به سمت جذب بیشتر مشتری پیش می‌برند (Verbruggen and Snoeck, 2023). شرکت مندیکس[1] از پیشگامان این عرصه، بیانیه‌ی سکوی کم‌کد شامل ۹ اصل را ارائه کرده است که عبارت‌اند از (۱) توسعه مدل‌رانده[2]، (۲) همکاری[3]، (۳) چابکی[4]، (۴) رایانش ابری، (۵) باز بودن[5]، (۶) توسعه چندکاربره[6]، (۷) آزمایش و نوآوری[7]، (۸) حاکمیت و کنترل[8] و (۹) جامعه[9] (Kenneweg et al., 2021).

مهم‌ترین اصل برای سکوی کم‌کد که بدون آن توسعه کم‌کد امکان‌پذیر نیست، توسعه مدل‌رانده است. توسعه مدل‌رانده به روش‌های ساخت و تولید نرم‌افزار با محوریت

1. https://www.mendix.com/
2. Model-Driven Development (MDD)
3. Collaboration
4. Agility
5. Openness
6. Multi-User Development
7. Experimentation and Innovation
8. Governance and Control
9. Community

مدل اطلاق می‌شود (Brambilla et al., 2017). همکاری به‌عنوان اصل دوم بین توسعه‌دهنده و متخصص دامنه وجود دارد که با استفاده از یک زبان بصری به‌درک مشترک از سیستم کمک کرده و نیازمند فناوری‌های هم‌زمانی و کنترل نسخه به‌صورت خودکار و درونی باشد تا اعضای تیم توسعه و متخصصین دامنه در هر لحظه و از هرجایی امکان بررسی و ویرایش مدل‌ها را داشته باشند (Brosch et al., 2009). روش‌های توسعه چابک چندین سال قبل از سکوی کم‌کد معرفی شدند (شهسواری پور و همکاران، ۱۳۹۶). چابکی و کم‌کد بودن هر دو یک هدف مشترک دارند: راهی بهتر برای توسعه نرم‌افزار (Kenneweg et al., 2021). اصل چهارم رایانش ابری موجب سهولت و سرعت در استقرار کاربردهای موردنیاز مشتری می‌شود و امکان استفاده از منابع و قابلیت‌های گسترده را جهت ساخت کاربردهای قدرتمند و هوشمند فراهم می‌کند (Rekstad, 2021). با توجه به اصل پنجم، سکوی توسعه برنامه‌های کاربردی برای یکپارچه کردن فناوری‌های متعدد باید باز باشد، یعنی برنامه‌های کاربردی باید به‌راحتی با سیستم‌های قدیمی موجود ادغام شوند و همچنین، بتوانند فناوری‌های جدید را که در آینده معرفی می‌شوند، در خود جای دهند (Bucaioni et al., 2022). اصل توسعه چندکاربره یعنی همه مصنوعات کد مانند کتابخانه‌ها، برنامه‌های افزودنی، مؤلفه‌های سفارشی و تمام منابعی که برنامه به آن نیاز دارد، در هرزمانی قابل‌دستیابی توسط همه کاربران مدل باشد (Kenneweg et al., 2021).

با در نظر گرفتن اصل هفتم، یک مزیت بزرگ سکوی کم‌کد این است که افراد غیرمتخصص را که تجربه نوشتن کد ندارند، در خلق و ایجاد برنامه‌های کاربردی مشارکت می‌دهد؛ درنتیجه این مزیت، می‌توان ایده‌هایی را آزمایش کرد که تأثیر قابل‌توجهی روی کسب‌وکار دارند (Gottschalk et al., 2022). با توجه به اصل حاکمیت و کنترل، سکوی کم‌کد مهندسی می‌شود تا به کاربران این امکان را بدهد که کارهای درست را به‌راحتی و کارهای نادرست را به‌سختی انجام دهند (Richardson and Rymer, 2014). سرانجام، در زیربنایی‌ترین سطح، وجود جامعه برای یک فناوری باعث می‌شود که آن را به انتخاب شغلی عالی برای توسعه‌دهندگان و انتخاب استراتژیک و اقتصادی

برای شرکت‌ها تبدیل کند. وقتی توسعه‌دهنده، جامعه‌ای فعال و بزرگ را ببیند، می‌تواند اطمینان داشته باشد که سرمایه‌گذاری او در یادگیری این فناوری نتیجه خواهد داد (Cabot, 2020).

برای بررسی ویژگی‌های سکوی توسعه کم‌کد از پژوهشی با عنوان پشتیبانی از درک و مقایسه سکوهای توسعه کم‌کد شش مرحله برای توسعه یک برنامه کاربردی معرفی شده است (Sahay et al., 2020). این مراحل عبارت‌اند از: (۱) مدل‌سازی دامنه، (۲) تعریف رابط کاربری، (۳) مشخصات منطق کسب‌وکار، (۴) یکپارچه‌سازی با خدمات خارجی، (۵) تولید و استقرار برنامه و (۶) نگهداشت (Sahay et al., 2020). در مرحله مدل‌سازی دامنه، ساختارهای مدل‌سازی برای نمایش مفاهیم و روابط زیربنای برنامه درحال‌توسعه به کاربران ارائه می‌شود (Guizzardi and Proper, 2021). در تعریف رابط کاربری، کاربران فرم‌ها و صفحات داده را برای ایجاد، ویرایش و تجسم داده‌هایی تعریف می‌کنند که قرار است توسط برنامه درحال‌توسعه مدیریت شوند (Akiki et al., 2014). در تعیین مشخصات کسب‌وکار، کاربران کنترل و جریان داده‌های سیستم درحال‌توسعه را به‌صورت بصری بیان می‌کنند (Harkes, 2019). در مرحله یکپارچه‌سازی با خدمات خارجی، سکوهای کم‌کد برای استفاده از خدمات و یا مصرف داده‌های ارائه‌شده توسط سیستم‌های شخص ثالث، با سرویس‌های خارجی و منابع داده یکپارچه می‌شوند (Bock and Frank, 2021). در مرحله تولید و استقرار، برنامه تولیدشده باید در زمان کوتاه در محیط‌های خصوصی یا عمومی مستقرشده و کاربر نهایی درگیر نصب و راه‌اندازی نشود (Wurster et al., 2020). سرانجام، در نگهداشت برنامه، نظارت و نگهداری به انجام می‌رسد (Bhattacharyya and Kumar, 2021).

پیشینه پژوهش

پژوهش‌های این بخش به دو دسته تقسیم می‌شوند. ابتدا، پژوهش‌هایی که به هر دو موضوع مهندسی نرم‌افزار مدل‌رانده و سکوی توسعه کم‌کد پرداخته‌اند، بررسی‌شده و سپس، پژوهش‌هایی که سکوهای توسعه کم‌کد معروف را معرفی کرده‌اند، مطالعه شده است.

پژوهش‌های دسته اول

در پژوهشی با عنوان موقعیت‌یابی جنبش کم‌کد در حوزه مهندسی مدل‌رانده بیان‌شده که رویکرد توسعه کم‌کد همان مهندسی مدل‌رانده است و صرفاً باهدف جذب مخاطب و درک بهتر نام رویکرد تغییر یافته است (Cabot, 2020). در پژوهش یادشده، سکوی کم‌کد به‌عنوان یک رویکرد محدودتر از مهندسی مدل‌رانده بوده، به‌طوری‌که نوع برنامه کاربردی (برنامه‌های تحت وب یا موبایل با داده‌های فشرده) هدف خواهد بود؛ همچنین، میزان نوشتن دستی کد راهی برای مقایسه دو رویکرد معرفی شده است (Cabot, 2020).

در پژوهشی با عنوان چالش‌ها و فرصت‌ها در آزمون کم‌کد بیان شده که سکوهای کم‌کد مبتنی بر مهندسی مدل‌رانده هستند (Khorram et al., 2020). در پژوهش ذکرشده، طراحی سیستم با مدل‌سازی بصری و تولید خودکار سیستم نهایی قابل‌اجرا به‌عنوان ویژگی مشترک توسعه کم‌کد و مهندسی مدل‌رانده معرفی شده است (Khorram et al., 2020).

در پژوهش چارچوب لوکوموت[1] بیان‌شده که سکوهای توسعه کم‌کد، سکوهای توسعه نرم‌افزار بر روی ابر هستند که می‌توان از آن‌ها به‌عنوان یک مدل رایانش ابری یعنی «سکو به‌عنوان یک خدمت»[2] یاد کرد (Tisi et al., 2020). در پژوهش چارچوب لوکوموت بیان‌شده، توسعه‌دهندگان می‌توانند از اصول مدل‌رانده بهره ببرند، اما همان‌طور که مقیاس‌پذیری یکی از مشکلات جدی در مهندسی مدل‌رانده است، در سکوی کم‌کد نیز این چالش نمود بیشتری دارد (Tisi et al., 2020).

در پژوهشی با عنوان مطالعه تجربی بحث‌های توسعه‌دهنده در مورد چالش‌های توسعه نرم‌افزار کم‌کد، بیان شده است که سکوی کم‌کد از مهندسی مدل‌رانده الهام گرفته و توسعه با استفاده از نمایش‌های انتزاعی به‌جای تمرکز بر محاسبات الگوریتمی، انجام می‌شود (Alamin et al., 2021). در پژوهش یادشده، هدف سکوی کم‌کد، کاهش پیچیدگی آزمایش، استقرار و نگهداری تعیین شده است (Alamin et al., 2021).

---

1. Lowcomote
2. Platform as a service

در پژوهشی با عنوان توسعه کم‌کد و مهندسی مدل‌رانده: دو روی یک سکه، با مقایسه سه رویکرد مهندسی مدل‌رانده، توسعه نرم‌افزار کم‌کد[1] و سکوی توسعه کم‌کد، پنج ناحیه پژوهشی طبقه‌بندی شده است (Ruscio et al., 2022). در پژوهش ذکرشده، تفاوت‌های میان مهندسی مدل‌رانده و سکوی کم‌کد ازجمله، کاربران نهایی و دامنه برنامه کاربردی بیان شده است (Ruscio et al., 2022).

پژوهش‌های دسته دوم

سکوی اپ‌میکر[2] توسط شرکت گوگل ارائه شده است. با این سکو می‌توان به‌راحتی انواع ابزارها و خدمات گوگل، ازجمله برنامه‌های داده‌محور را بالادست پایگاه داده ابری گوگل توسعه داد (Käss et al., 2023). در سال ۲۰۲۱، این سکو توسط اپ‌شیت[3] خریداری شد و در حال حاضر با این نام شهرت دارد (Bucaioni et al., 2022).

سکوی ارائه‌شده توسط اپل، سوییفت‌یوآی[4] نام دارد که با عبارت «برنامه‌های کاربردی بهتر، کد کمتر» معرفی شد (Nekras, 2022). با این برنامه می‌توان با کمک نحو اعلانی، پشتیبانی از قابلیت کشیدن و رها کردن[5] و پیش‌نمایش در زمان واقعی (رویکرد ویزی‌ویگ[6]: آنچه می‌نگری همان است که درنهایت به‌دست می‌آوری) رابط کاربری را برای هر دستگاه اپل ایجاد کرد (Nekras, 2022).

اخیراً، آمازون از انتشار سکوی بدون‌کد، هانی‌کد[7] خبر داد. در اصل، این سکو یک مؤلفه‌ی صفحه گسترده و مجموعه‌ای از قالب‌ها برای ایجاد برنامه‌های ساده مبتنی بر وب می‌باشد (ElBatanony and Succi, 2021).

پاوراپس[8] سکوی بدون‌کد است که توسط مایکروسافت جهت استفاده‌ی کاربران

1. Low-Code Software Development
2. App Maker
3. App Sheet: https://www.appsheet.com/
4. SwiftUI: https://developer.apple.com/xcode/swiftui/
5. Drag and Drop
6. WISWYG (what you see is what you get)
7. HoneyCode: https://www.honeycode.aws/
8. PowerApps: https://powerapps.microsoft.com/en-us/

تجاری ارائه شده است که با مدل داده و فرآیندهای تجاری شروع‌شده و تا تولید خودکار برنامه‌های کاربردی همه‌جانبه و پاسخگوی قابل‌حمل پیش می‌رود. (Gürcan and Taentzer, 2021).

اوت‌سیستمز[1] سکوی توسعه کم‌کدی است که امکان توسعه برنامه‌های دسکتاپ و موبایل را فراهم می‌کند. اوت‌سیستمز دو جز مهم دارد. اول اینکه یک استودیوی میانی برای اتصال به پایگاه داده از طریق دات نت یا جاوا دارد. دوم، یک استودیوی خدماتی برای تعیین رفتار برنامه درحال‌توسعه دارد (Martins et al., 2020).

مندیکس[2] سکوی توسعه کم‌کدی که ابداً نیازی به کدنویسی ندارد و می‌توان به همه ویژگی‌ها از طریق قابلیت کشیدن و رها کردن دسترسی پیدا کرد. این سکو اصل دوم (همکاری) از بیانیه‌ی توسعه کم‌کد را به‌خوبی پیاده‌سازی کرده است (Gürcan and Taentzer, 2021).

کیس‌فلو[3] سکوی نرم‌افزار اتوماسیون گردش کار مبتنی بر ابر است که به کاربران کمک می‌کند تا برنامه‌های کاربردی سازمانی خودکار را ایجاد و اصلاح کنند. اهداف اصلی آن برنامه‌های کسب‌وکار کوچک با ویژگی‌های عملکردی کامل است که برای استفاده داخلی و گردش‌های کاری انسان‌محور ضروری هستند (Hili and Oliveira, 2022).

ایپیان[4] یکی از قدیمی‌ترین سکوهای توسعه کم‌کد است که امکان ایجاد برنامه‌های کاربردی موبایل و وب را از طریق ابزار شخصی‌سازی، ابزارهای داخلی همکاری تیمی، مدیریت کار و شبکه اجتماعی فراهم می‌کند (Vincent et al., 2019).

روش‌شناسی پژوهش

روش پژوهش حاضر، ازنظر هدف کاربردی و به لحاظ ماهیت توصیفی- پیمایشی است.

1. OutSystems: https://www.outsystems.com/
2. Mendix
3. Kissflow: https://kissflow.com/
4. Appian: https://appian.com/

هدف از این پژوهش، بررسی رویکردهای متنوع در زمینه سکوهای توسعه کم‌کد و مقایسه آن با رویکردهای مهندسی مدل‌رانده می‌باشد. بر این اساس، ابتدا سه پرسش پژوهش طراحی شد. پرسش اول بررسی ویژگی‌های سکوهای توسعه کم کد است. برای پاسخ به پرسش اول، جهت جمع‌آوری اطلاعات راجع به سکوهای کم‌کد به دو طریق مطالعات انجام شده است: (۱) بررسی مقالات از ۲۰۱۴ تا ۲۰۲۳ در پایگاه داده‌های گوگل اسکلار[1]، دی‌بی‌ال‌پی[2] و ریسرچ‌گیت[3] و (۲) بررسی ابزارهای کاربردی توسعه کم‌کد. برای فیلتر کردن مقالات، معیارهایی از قبیل بیشترین ارتباط، مرتبط بودن با زمینه «مهندسی مدل‌رانده» و استفاده از کلیدواژه‌های مرتبط با مدل‌رانده، سال انتشار و کاربردی بودن به‌لحاظ معرفی یک ابزار یا نرم‌افزار استفاده شد. برای بررسی ابزارهای کم‌کد دو معیار در نظر گرفته شد؛ اولاً، ابزار مدنظر در یک مقاله در گوگل اسکولار معرفی شده باشد؛ ثانیاً، ابزار مدنظر دارای پورتال آنلاین بوده تا بتوان ویژگی‌های آن را با دقت بررسی نمود. پرسش دوم، بررسی جایگاه سکوهای توسعه کم‌کد در مقایسه با رویکرد مهندسی مدل‌رانده است. با توجه به این‌که رویکرد توسعه کم‌کد از سال ۲۰۱۴ مطرح شده است، مهندسی مدل‌رانده حوزه گسترده‌تری از تحقیقات را به‌خود اختصاص داده است. بررسی مقالات از سال ۲۰۱۴ به بعد بوده و مقالاتی که به هر دو موضوع پرداخته‌اند، موردتوجه قرار گرفته است. همچنین، با توجه به نقش رایانش ابری، برای طراحی مدل توصیفی جایگاه و پاسخ به پرسش دوم، بررسی شد که مقالات به مفهوم رایانش ابری نیز پرداخته باشند. با توجه به اهمیت پژوهش و نبود خبره در زمینه توسعه کم‌کد در ایران، سعی شد مقالاتی که نویسندگان معتبر در حوزه مدل‌رانده دارند، موردتوجه قرار گیرند. این مقالات، به‌طورمعمول با اصطلاح «صدای متخصص[4]» از مقالات پژوهشی مجزا می‌شوند. چالش اصلی برای سکوهای توسعه کم‌کد، تجاری بودن این سکوهاست که باعث می‌شود جامعه محدودی از کاربران بتوانند از آن استفاده کنند. ابزارهای مهندسی مدل‌رانده اغلب

1. https://scholar.google.com/
2. https://dblp.uni-trier.de/
3. https://www.researchgate.net/
4. Expert Voice

دانشگاهی و رایگان هستند. درنتیجه، پرسش سوم به بررسی الزامات برای انتقال از مهندسی مدل‌رانده به توسعه کم‌کد پرداخت. برای پاسخ به این پرسش، یک نمونه موردمطالعه معرفی شد و به‌صورت فنی بررسی شد که چه نیازمندی‌هایی لازم است تا یک نمونه از زبان مدل‌سازی در حوزه مهندسی مدل‌رانده به سکوی توسعه کم‌کد مهاجرت کند.

یافته‌ها

در این بخش به سؤالات پژوهش پاسخ داده می‌شود.

پاسخ به پرسش اول پژوهش

طبق مطالعات انجام‌شده، از بین هشت سکوی توسعه کم‌کد معرفی‌شده در بخش پیشینه پژوهش، سکوی ارائه‌شده توسط اپل به‌نام سوییفت‌یوآی مبتنی بر ۹ اصل توسعه کم‌کد نمی‌باشد و در مقایسه‌های بعدی حذف شده است. برای بررسی دقیق‌تر ویژگی‌های سکوها، شش مرحله معرفی‌شده در مبانی نظری پژوهش در نظر گرفته شده است. با مطالعه سکوهای توسعه کم‌کد، ویژگی‌های جزئی‌تر هر مرحله شناسایی شده است. به‌عنوان نمونه، برای مرحله رابط کاربری قابلیت‌های متعددی مانند کشیدن و رها کردن، نقطه و کلیک، گزارش‌ها و داشبوردهای از پیش ساخته‌شده، گزارش‌گیری پیشرفته، گردش کار داخلی و قابل تنظیم و استفاده از فرم‌ها را می‌توان ازجمله ویژگی‌های جزئی‌تر دانست.

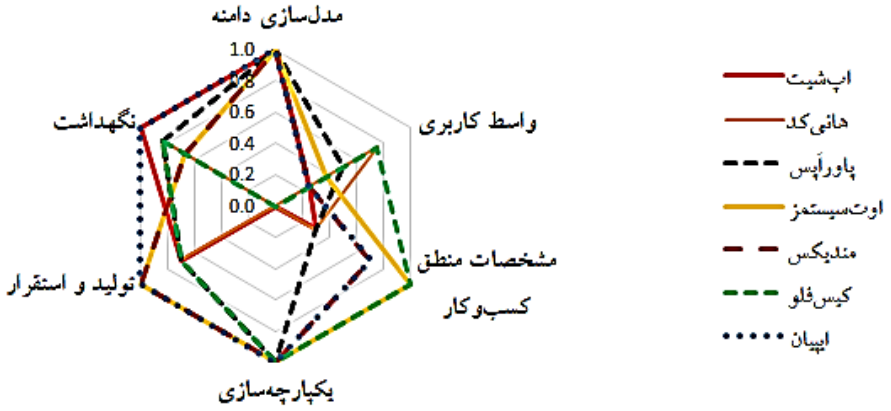
در جدول ۱ مراحل و ویژگی‌های هفت سکوی توسعه کم‌کد بررسی و نمایش داده شده است. برای مثال دو ابزار (هانی‌کد و کیس‌فلو) از هفت ابزار از مدل‌سازی دامنه پشتیبانی نمی‌کند.

جدول ۱. مقایسه ویژگی‌های هفت سکوی توسعه کم‌کد، منبع: یافته‌های پژوهش حاضر.

ویژگی‌ها	اپ‌شیت	هانی کد	پاورآپس	اوت‌سیستمز	مندیکس	کیس‌فلو	اپیان
مدل‌سازی دامنه	✓	✗	✓	✓	✓	✗	✓
واسط کاربری							
قابلیت کشیدن و رها کردن	✓	✓	✗	✓	✓	✓	✓
گزارش‌های از پیش ساخته‌شده	✓	✓	✓	✓	✓	✓	✓
داشبوردهای از پیش ساخته‌شده	✗	✓	✓	✓	✗	✓	✗
گزارش‌گیری پیشرفته	✗	✓	✗	✗	✗	✓	✗
گردش کار داخلی	✗	✓	✗	✗	✗	✓	✗
گردش کار قابل تنظیم	✗	✓	✗	✗	✗	✓	✗
رویکرد نقطه و کلیک	✗	✗	✓	✗	✗	✗	✗
استفاده از فرم‌ها	✗	✗	✓	✗	✗	✗	✗
مشخصات منطق کسب‌وکار							
موتور قوانین کسب‌وکار	✓	✓	✓	✓	✓	✓	✓
ویرایشگر گردش کار گرافیکی	✗	✗	✗	✓	✓	✓	✗
منطق کسب‌وکار هوشمند	✗	✗	✗	✓	✗	✓	✓
یکپارچه‌سازی	✗	✗	✓	✓	✓	✓	✓
تولید و استقرار							
اجرای مدل	✓	✓	✓	✓	✓	✓	✓
استقرار ابری	✓	✓	✓	✓	✓	✓	✓
استقرار محلی	✗	✗	✗	✓	✓	✗	✓
نگهداشت							
نظارت بر رویداد	✓	✓	✓	✓	✓	✓	✓
خودکارسازی فرایند	✓	✓	✓	✓	✗	✓	✓
کنترل فرآیند تأیید	✓	✓	✗	✗	✗	✓	✓
مدیریت موجودی	✓	✓	✓	✓	✓	✗	✓
مدیریت کیفیت	✓	✗	✓	✗	✓	✓	✓
مدیریت گردش کار	✓	✓	✓	✓	✓	✓	✓

بر اساس مراحل و ویژگی‌های ارائه‌شده در جدول ۱ و مقایسه‌ای که روی ابزارهای مختلف انجام شده است، نمودار شکل ۱ طراحی شد. درواقع، هر چه سطح بیشتری از نمودار توسط ابزار پوشانده شود، پشتیبانی بهتری از سکوی توسعه کم‌کد توسط ابزار صورت گرفته است. ابزار کیس‌فلو در مقایسه با ابزارهای دیگر ویژگی‌های بیشتری از توسعه کم‌کد را ارائه می‌دهد.

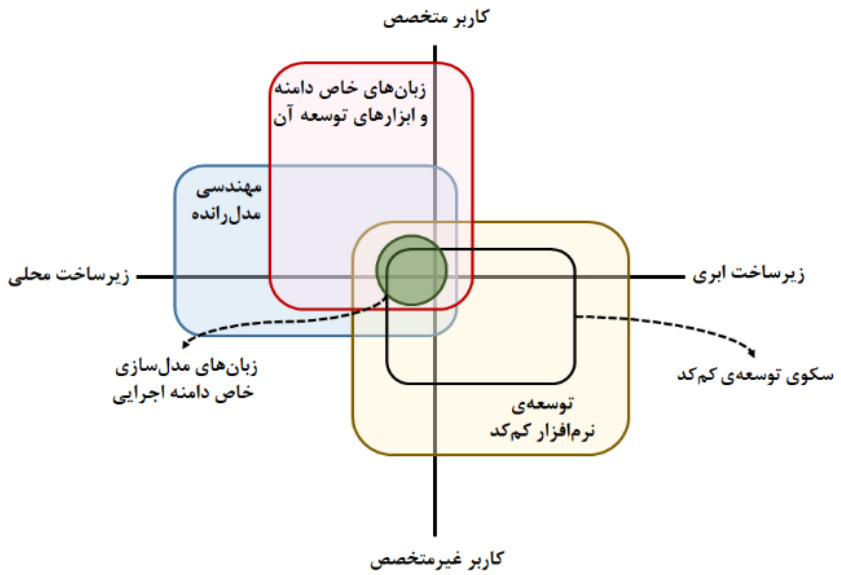
شکل ۱. مقایسه ابزارهای توسعه کم‌کد بر اساس ویژگی‌های پیشنهادی، منبع: یافته‌های پژوهش حاضر.



پاسخ به پرسش دوم پژوهش

برای بررسی جایگاه سکوهای توسعه کم‌کد در مقایسه با رویکردهای مهندسی مدل‌رانده، شکل ۲، بر اساس مطالعات بررسی‌شده در پیشینه پژوهش به‌دست آمده است. شکل ۲، جایگاه سکوی کم‌کد را در مقایسه با رویکردهای مطرح نشان می‌دهد. در محور افقی مفهوم زیرساخت هم برای توسعه و هم برای استقرار درنظر گرفته شده است. در محور عمودی کاربر هم ازنظر توسعه و هم ازنظر استفاده، نمایش داده شده است.

شکل ۲. جایگاه سکوی توسعه کم‌کد در مقایسه با مهندسی مدل‌رانده، منبع: یافته‌های پژوهش حاضر.



با توجه به شکل ۲، سکوی توسعه کم‌کد نوعی روش توسعه نرم‌افزار با میزان کد دستی کمینه است. توسعه نرم‌افزار کم‌کد[1] مفهومی کلی‌تر از سکوی کم‌کد بوده و می‌تواند روش‌ها و ابزارهای خودکارسازی را نیز شامل شود. ویژگی زیرساخت ابری این امکان را برای کاربران فراهم می‌کند که بدون صرف وقت جهت راه‌اندازی و نصب سکو بتوانند از آن استفاده کنند. اکثر سکوهای شناخته‌شده، (مطابق جدول ۱) مبتنی بر ابر هستند و بر طبق ۹ اصل اشاره‌شده در مبانی نظری پژوهش، رایانش ابری یکی از ملزومات این نوع سکوهاست؛ اما برخی از سکوهای توسعه کم‌کد می‌توانند در سازمان مشتری جهت استفاده مستقر شوند. از طرف دیگر، سکوهای توسعه کم‌کد باید قابلیت‌هایی را جهت استقرار برنامه کاربردی تولیدشده فراهم آورند. معمولاً برای این‌که کاربر نهایی درگیر نصب برنامه نشود، استقرار به‌صورت ابری در این سکوها پیشنهاد می‌شود.

توسعه‌دهندگانی که از سکوی کم‌کد برای تولید برنامه کاربردی استفاده می‌کنند،

1. Low-Code Software Development (LCSD)

تخصص برنامه‌نویسی کم و محدودی دارند و یا دارای تجربه فنی کمی در زمینه توسعه نرم‌افزار هستند. اغلب روش‌های توسعه کم‌کد برای این دسته از کاربران ارائه شده‌اند. بدین دلیل، دو حوزه‌ی توسعه کم‌کد و سکوی کم‌کد در بخش پایینی نمودار قرار داده شده است.

در مقابل، اکثر رویکردهای مهندسی مدل‌رانده از زیرساخت محلی و ابزارهای دسکتاپی مانند چارچوب مدل‌سازی اکلیپس[1] استفاده می‌کنند و توسعه‌دهندگان و کاربران مهندسی مدل‌رانده اغلب مهندسین نرم‌افزار و برنامه‌نویسان هستند. همان‌طور که نشان داده شده، همه روش‌های مهندسی مدل‌رانده، توسعه کم‌کد نیستند و بالعکس. به‌عبارت‌دیگر، بخشی از روش‌های مهندسی رفت‌وبرگشت باهدف کاهش کدنویسی دستی ارائه نشده‌اند و نمی‌توانند روش توسعه کم‌کد محسوب شوند. همچنین، روش‌های دیگر مدل‌رانده، اگرچه باهدف کاهش میزان کدنویسی دستی ارائه شده‌اند، اما نمی‌توان نام سکو را برای آن‌ها اطلاق کرد. درواقع، در این روش‌ها به‌دلیل نبود زیرساخت ابری، کاربر وقت زیادی را صرف نصب و راه‌اندازی ابزار مدل‌رانده می‌کند و برنامه تولیدشده در برخی اوقات به‌دلیل نقص، قابلیت اجرایی ندارد. درنتیجه، استقرار آن به‌صورت خودکار امکان‌پذیر نیست.

زبان‌های خاص دامنه در بخش بالایی نمودار قرار دارد که نشان می‌دهد، کاربران زبان‌های خاص دامنه نیز برنامه‌نویسان هستند. برخی از زبان‌های خاص دامنه که برپایه‌ی اصول مدل‌رانده توسعه‌یافته‌اند، جز رویکرد مهندسی مدل‌رانده هستند؛ اما برخی دیگر از زبان‌ها و ابزارهای توسعه زبان‌ها مشترک با این حوزه نیستند، برای مثال برخی ابزارها قابلیت تولید پارسر از روی گرامر تعریف‌شده برای زبان مدنظر را دارند، اما نمی‌توانند جز ابزارهای توسعه زبان‌های خاص دامنه قرار گیرند.

وجه اشتراک زبان‌های خاص دامنه و مهندسی مدل‌رانده را می‌توان «زبان‌های مدل‌سازی خاص دامنه[2]» نامید. زبان‌های مدل‌سازی خاص دامنه امکان مدل‌سازی مفاهیم

1. https://www.eclipse.org/modeling/emf/
2. Domain-Specific Modeling Language

یک قلمروی خاص را فراهم می‌کنند. درصورتی‌که برای این زبان‌ها به‌غیر از مؤلفه‌های نحو انتزاعی[1] (فرامدل[2]) و نحو عینی[3] (شکل متنی یا گرافیکی)، مؤلفه‌ی معنای پویا[4] هم تعریف شده باشد، زبان مدل‌سازی قابلیت اجرایی پیدا می‌کند (Samimi-Dehkordi et al., 2019). همان‌طور که در شکل ۱ نشان داده‌شده، اکثر زبان‌های مدل‌سازی خاص دامنه قابل‌اجرا را می‌توان جز سکوی توسعه کم‌کد در نظر گرفت. البته برخی از این زبان‌ها با توجه به این‌که مبتنی بر ابر نیستند و تمهیداتی برای استقرار خروجی تولیدشده را در خود ندارند، سکوی کم‌کد نبوده و فقط یک روش توسعه کم‌کد در نظر گرفته می‌شوند. معمولاً کاربران این زبان‌ها با توجه به نحو عینی زبان ممکن است، افراد متخصص یا غیرمتخصص در زمینه برنامه‌نویسی و مهندسی نرم‌افزار باشند.

پاسخ به پرسش سوم پژوهش

سکوی توسعه کم‌کد یک رویکرد نوظهور است که به‌دلیل استفاده از رایانش ابری قابلیت نصب سریع و راحت، حمل‌پذیری بالا، مقیاس‌پذیری، کارایی بالا و هزینه‌ی کم توسعه را امکان‌پذیر می‌سازد (Käss et al., 2023). مهاجرت به سمت ابر دارای مزایای زیادی است؛ استفاده از سرویس‌های ابری، منجر به صرفه‌جویی زیادی در زمان می‌شود که می‌توان آن را صرف پرداختن به سایر مسائل و موضوعات فنی در سازمان کرد؛ به‌علاوه، برای کاربران هزینه‌ای بابت سخت‌افزار وجود ندارد. رایانش ابری باعث افزایش مقیاس‌پذیری می‌شود و از هر مکانی با هر دستگاهی که به اینترنت متصل باشد، قابل‌دسترس است؛ کاربر برای شروع هزینه جداگانه‌ای ندارد و به‌روزرسانی نرم‌افزارها به‌صورت خودکار انجام می‌شود. نرم‌افزارها می‌توانند برای هر مشتری، خاص او باشد و تنظیمات خاص داشته باشد. در آخر، راه‌اندازی سریع و بدون نیاز به نیروی فنی و متخصص انجام می‌گیرد (Wei and Blake, 2010)؛ بنابراین، جامعه مدل‌رانده نیز در تلاش

1. Abstract Syntax
2. Metamodel
3. Concrete Syntax
4. Dynamic Semantics

برای مهاجرت به سمت ارائه نسخه‌های ابری برای ابزارهای پرکاربرد خود می‌باشد (Sahay et al., 2020).

انتقال یک زبان مدل‌سازی از رویکرد مهندسی مدل‌رانده به رویکرد سکوی کم‌کد، انتقال یک فناوری به سمت ابر می‌باشد (Zhao and Zhou, 2014). طبق شکل ۲، در صورت انتقال، زبان‌های مدل‌سازی خاص دامنه مبتنی بر ابر بیشتری را می‌توان توسعه داد که به‌عنوان سکوهای کم‌کد خاص دامنه، می‌توانند معرفی شوند. از نظر انتقال فناوری‌ها به یک سناریوی مبتنی بر ابر، می‌توان تمام کارهای موجود را به سه دسته طبقه‌بندی کرد: (۱) مفاهیم، (۲) ویژگی‌های بی‌سر[1] و (۳) ویژگی‌های رابط کاربری (Rekstad, 2021). برای مثال، با در نظر گرفتن یک زبان مدل‌سازی، نحو انتزاعی به‌عنوان یک مفهوم، معنای زبان در قالب برنامه تبدیل به‌عنوان یک ویژگی بی‌سر و نحو عینی به‌صورت یک ویژگی رابط کاربری می‌باشد. مفاهیم معمولاً مخصوص به دسکتاپ یا وب نیستند، بنابراین می‌توان آن‌ها را همان‌طور که هستند، دوباره استفاده کرد. ویژگی‌های بی‌سر اغلب می‌توانند در زمینه ابری نیز استفاده شوند. بااین‌حال، برای در دسترس قرار دادن ویژگی‌های بی‌سر موجود در فضای ابری، باید یک وب‌سرویس[2] وجود داشته باشد. درنهایت، ویژگی‌های رابط کاربری نمی‌توانند به‌سادگی به مرورگر منتقل شوند؛ بنابراین، برخی چیزها باید در آنجا دوباره اجرا شوند.

طبق مطالعات انجام‌شده، برای مهاجرت یک زبان مدل‌سازی به سمت ابر، باید اجزای زبان موردبررسی قرار گیرد. اجزای یک زبان مدل‌سازی عبارت‌اند از: (۱) نحو انتزاعی، (۲) نحو عینی و (۳) معنای پویا (Samimi-Dehkordi et al., 2019). با توجه به این‌که نحو انتزاعی شامل مفاهیم زبان و ارتباط بین مفاهیم است که معمولاً در قالب یک فرامدل طراحی می‌شود، مستقل از دستکتاپ یا وب بوده و می‌تواند در بستر وب استفاده شود؛ اما نحو عینی، شکل ظاهری مفاهیم را بیان می‌کند؛ یعنی، همان رابط کاربری زبان خواهد بود و باید از اول پیاده‌سازی شود. درنهایت، معنای پویا به روش ترجمه‌ای (یعنی استفاده از

1. Headless features
2. Webservice

یک تولیدکننده‌ی کد)، یک ویژگی بی‌سر است که برای اجرا نیاز به وب‌سرویس دارد تا برنامه‌های نوشته‌شده را فراخوانی کند.

برای تکمیل پاسخ به پرسش سوم، یک نمونه ساده جهت بررسی نحوه مهاجرت یک زبان مدل‌سازی به ابر ارائه می‌شود. نمونه موردمطالعه مرتبط با قرارداد هوشمند[1] است. «قراردادهای هوشمند» اصطلاحی است که برای توصیف کد کامپیوتری استفاده می‌شود که به‌طور خودکار تمام یا بخش‌هایی از یک توافق‌نامه را اجرا می‌کند و در یک سکوی مبتنی بر زنجیره بلوک[2] ذخیره می‌شود (Zheng et al., 2020). برای انتخاب نمونه، یک زبان مدل‌سازی در رویکرد مدل‌رانده انتخاب شد. این نمونه از پژوهشی با نام «سپ‌چین[3]: یک راه‌حل مدل‌رانده گرافیکی برای یکپارچه‌سازی پردازش رویدادهای پیچیده و زنجیره بلوک» انتخاب شد (Boubeta-Puig et al., 2021). در فرامدل پیشنهادشده، شش فراکلاس[4] و یک نوع شمارشی[5] وجود دارد. فراکلاس‌ها عبارت‌اند از: ریشه[6]، قرارداد هوشمند[7]، تابع قرارداد[8]، پارامتر[9]، پارامتر ورودی[10] و پارامتر خروجی[11]. فراکلاس ریشه از مجموعه‌ای از یک یا چند قرارداد هوشمند برای یک دامنه کاربردی تشکیل شده است. برای فراکلاس ریشه، صفاتی از قبیل نام، توضیحات متنی و تاریخ ایجاد مشخص شده است. فراکلاس قرارداد هوشمند کلیدی است زیرا حاوی حساس‌ترین اطلاعات است و تنها فراکلاسی است که می‌تواند توسط رویدادهای پیچیده مرتبط شود. هر قرارداد هوشمند دارای یک نام، مسیر تصویر، کلید خصوصی و آدرس قرارداد است. هر قرارداد هوشمند از یک یا چند تابع قرارداد تشکیل شده است. هر تابع شامل یک یا چند پارامتر

1. Smart Contract
2. Block Chain
3. CEPchain
4. MetaClass
5. Enumeration
6. Root
7. SmartContract
8. ContractFunction
9. Parameter
10. InputParameter
11. OutputParameter

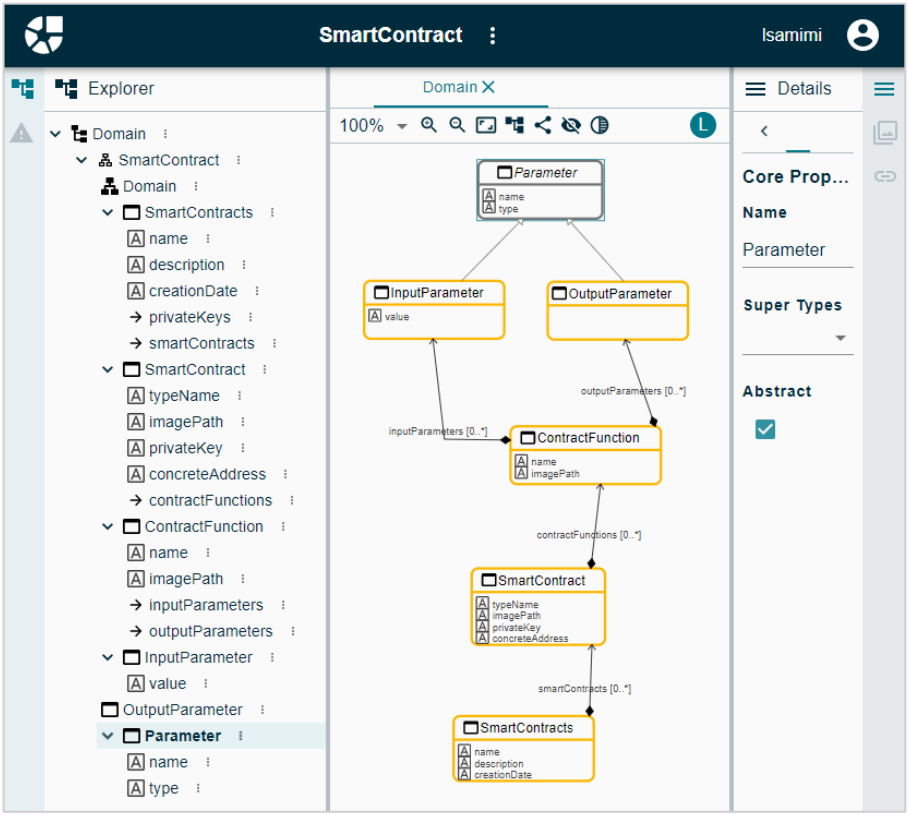
ورودی و یک پارامتر خروجی است. فراکلاس پارامتر انتزاعی است که ویژگی‌های مشترک برای انواع مختلف پارامترها را تعریف می‌کند (Boubeta-Puig et al., 2021). پس از تعریف فرامدل، نحو عینی و معنا توصیف شده است. برای نوشتن نحو عینی و معنا از چارچوب اپسیلون[1] استفاده شده است (Kolovos et al., 2006).

برای مهاجرت زبان مدل‌سازی معرفی‌شده به سمت ابر، لازم است از یک سکوی مستقر بر ابر استفاده شود. سکوی وب‌سیریوس[2] یک سکوی برخط مبتنی بر ابر است (Bedini et al., 2021). با توجه به این‌که نحو انتزاعی یک مفهوم مستقل از سکو بوده درنتیجه، مفاهیم همانند قبل خواهد بود؛ اما برای آن‌که بتوان نحو عینی را پیاده کرد، لازم است مفاهیم در سکوی وب‌سیریوس پیاده شود. برای این منظور لازم است ابتدا یک حساب کاربری در این سکو ایجاد شود و سپس، پروژه با نام دلخواه ایجاد شود. در محیط پروژه برای هر مفهوم، یک فراکلاس ایجاد می‌شود. فرامدل ساخته‌شده در شکل ۳ نمایش داده شده است. این محیط قابلیت تعریف نحو عینی گرافیکی را فراهم کرده است. برای تولید کد می‌توان از چارچوب اپسیلون که در نمونه قبلی استفاده شده بود، همچنان استفاده کرد.

بر طبق یافته‌ها، به‌دلیل نوپا بودن مفهوم سکوی توسعه کم‌کد و کمبود ابزارهای مهندسی مدل‌رانده مبتنی بر وب، ابزار وب‌سیریوس در مقایسه با ابزارهای رویکرد مدل‌رانده بلوغ لازم را ندارد. برای مثال، نوع صفات درون هر فراکلاس محدود است و امکان تعریف رابطه‌های دوطرفه بین فراکلاس‌ها در ابزار سیریوس وب وجود ندارد. در مقابل، برای تعریف و پیاده‌سازی فرامدل نیاز به نصب ابزار وجود ندارد و صرفاً با ورود به فضای ابری می‌توان مدل را ترسیم نمود.

1. Epsilon
2. SiriusWeb

شکل ۳. نمایی از پیاده‌سازی تطبیق زبان مدل‌سازی قرارداد هوشمند با سکوی توسعه کم‌کد، منبع: یافته‌های پژوهش حاضر.



بحث و نتیجه‌گیری

در این مقاله، به‌معرفی رویکرد نوظهور سکوی توسعه کم‌کد پرداخته شد. این سکوها باهدف کم کردن میزان کدنویسی دستی، تحویل سریع برنامه کاربردی به بازار و استفاده از پتانسیل توسعه‌دهندگان غیرمتخصص با دانش محدود برنامه‌نویسی به وجود آمدند. برای آن‌که اهداف توسعه کم‌کد محقق شود، استفاده از رایانش ابری ضروری است. در این مقاله، ابتدا ویژگی‌های مهم سکوهای توسعه کم‌کد بررسی شد و هفت ابزار مختلف بر اساس ویژگی‌های بررسی‌شده مورد مقایسه قرار گرفت. همچنین، به‌دلیل وجوه مشترک این حوزه با رویکرد مهندسی مدل‌رانده و با توجه به اصل اول بیانیه ۹ بندی توسعه کم‌کد،

مقایسه‌ای بین این دو حوزه ارائه شد و جایگاه سکوهای توسعه کم‌کد مشخص گردید. برای این‌که مهندسی مدل‌رانده بتواند از مزایای توسعه کم‌کد بهره‌مند شود، مهاجرت زبان‌های مدل‌سازی از رویکرد مدل‌رانده به سکوی توسعه کم‌کد موردمطالعه قرار گرفت و یک نمونه مهاجرت از زبان مدل‌سازی قراردادهای هوشمند به سکوی کم‌کد بررسی شد. از محدودیت‌های مهم این پژوهش، کمبود منابع کافی در حوزه توسعه کم‌کد به‌دلیل جدید بودن این حوزه بوده و همچنین، اغلب سکوهای رایج تجاری بوده و تعداد کمی سکوی رایگان در حال حاضر وجود دارد. بدین منظور، استفاده از تجربیات حوزه مدل‌رانده در توسعه این نوع سکوها پیشنهاد شد.

درس‌هایی که دو حوزه سکوی کم‌کد و مهندسی مدل‌رانده می‌توانند از یکدیگر فراگیرند، در ادامه به‌صورت مختصر بیان می‌شود. اولین موضوع دامنه برنامه‌های کاربردی است. اغلب سکوهای کم‌کدی که تا به امروز پیشنهاد شده است، تلاش کرده‌اند گستره‌ی وسیعی از کاربردها را پوشش دهند. درحالی‌که مهندسی مدل‌رانده به این نتیجه رسیده است که برای بهبود کیفیت برنامه‌های کاربردی، زبان‌های مدل‌سازی را مخصوص به یک دامنه توسعه دهد. این موضوع باعث می‌شود همه مفاهیم یک دامنه خاص در زبان دیده شود و کد تولیدشده دقیق‌تر و کامل‌تر باشد؛ بنابراین، سکوهای کم‌کد نیز بهتر است خاص یک دامنه طراحی شوند. دومین موضوع مهم، بسترهای مبتنی بر ابر و وب است. یکی از امتیازات اصلی که جامعه فنی و علمی را به سمت سکوهای کم‌کد جذب کرده است، عدم نیاز این سکوها به نصب و راه‌اندازی است. این امر به‌دلیل طراحی این سکوها بر بستر ابر می‌باشد. در مقابل، مهندسی مدل‌رانده در این دو دهه نتوانسته با چنین استقبال گسترده‌ای روبرو شود؛ بنابراین، لازم است مهندسی مدل‌رانده، ابزارها و روش‌هایی را ارائه دهد تا بتوان زبان‌های مدل‌سازی را بر بستر وب توسعه داد و کاربران درگیر نصب نسخه‌های نرم‌افزارهای دسکتاپی نشوند. موضوع سومی که اهمیت دارد، متن‌باز بودن اکثر ابزارهای مهندسی مدل‌رانده است. این موضوع باعث شده تا یادگیری و تکامل در روش‌های مهندسی مدل‌رانده به‌راحتی انجام پذیرد. در مقابل، اکثر سکوهای کم‌کد تجاری بوده و

متن‌باز نیستند. لازم است حوزه سکوی کم‌کد همانند مهندسی مدل‌رانده ابزارهای متن‌باز را ارائه دهد تا تکامل در این حوزه سریع‌تر پیش برود. مهندسی مدل‌رانده در تلاش است با حرکت به سمت وب، این بستر را برای توسعه کم‌کد فراهم نماید. از محدودیت‌های پژوهش حاضر، عدم دسترسی به سکوهای کم‌کد تجاری بوده است.

به‌عنوان کارهای آینده، لازم است ویژگی‌ها و پیچیدگی برنامه‌های کاربردی با استفاده از سکوی توسعه کم‌کد ساخته شده‌اند، باهدف ارزیابی وضعیت عملکرد این سکوها و استدلال در مورد معیارهایی مانند مقیاس‌پذیری و کارایی آن‌ها مطالعه شود؛ زیرا مانند هر فناوری دیگری، اگرچه توسعه کم‌کد مجموعه‌ای از مزایا را برای برنامه‌نویسان و توسعه‌دهندگان فراهم می‌کند، اما در هنگام استفاده از این رویکرد با چالش‌هایی مواجه خواهد شد. در برخی موارد، توسعه‌دهندگان از استفاده از سکوهای کم‌کد اجتناب می‌کنند و برنامه‌نویسی سنتی را ترجیح می‌دهند. مقیاس‌پذیری محدود، خطرات امنیتی و مشکلات یکپارچه‌سازی برنامه‌های توسعه‌یافته برخی از این چالش‌ها هستند. همچنین، مطالعات و ارزیابی‌های بیشتری باید توسط محققان برای یافتن و پیشنهاد راه‌حل‌های مناسب برای مسائل سکوهای کم‌کد انجام شود. پرداختن به این مسائل ممکن است به بهبود تجربیات توسعه‌دهندگان و برنامه‌نویسان کمک کند. مطالعات بیشتر می‌تواند کسانی را که از سکوهای کم‌کد استفاده نمی‌کنند، برای شروع استفاده از آن ترغیب کند. توسعه‌دهندگان باید انعطاف بیشتری برای سفارشی کردن عملکردهای داخلی در این سکوها داشته باشند. به‌علاوه، افزایش سطح امنیت برنامه‌های توسعه‌یافته با این سکوها باید توسط ارائه‌دهندگان موردتوجه قرار گیرد. همچنین، کد تولیدشده به‌صورت خودکار باید خواناتر باشد، با نظرات واضح نوشته شود و از نام متغیرهای معنی‌دار استفاده کند. این کار حفظ و تغییر کد را آسان می‌کند.

سپاسگزاری

مقاله حاضر مستخرج از نتایج طرح تحقیقاتی اجراشده به شماره قرارداد ۵۴۹۷/۱۴۱ از محل اعتبارات معاونت پژوهش و فناوری دانشگاه شهرکرد می‌باشد.

تعارض منافع

تعارض منافع ندارم.

ORCID
Leila Samimi-Dehkordi **http://orcid.org/0000-0002-2842-0256**
Abbas Horri **http://orcid.org/0000-0002-7105-1644**

منابع

شهسواری پور، ناصر؛ رضوان‌دوست، شهلا؛ میرزایی، امیر؛ حیدربیگی، شهلا. (۱۳۹۶). رابطه بین همسویی راهبرد فناوری اطلاعات و راهبرد کسب‌وکار با چابکی سازمانی در شرکت‌های نرم‌افزاری. مطالعات مدیریت کسب‌وکار هوشمند، ۵(۱۹)، ۷۵-۱۰۳. doi: 10.22054/ims.2017.7055

References

Akiki, P. A., Bandara, A. K., & Yu, Y. (2014). Adaptive model-driven user interface development systems. *ACM Computing Surveys (CSUR)*, 47(1), 1-33. https://doi.org/10.1145/2597999

Alamin, M., Malakar, S., Uddin, G., Afroz, S., Haider, T., Iqbal, A. (2021). An Empirical Study of Developer Discussions on Low-Code Software Development Challenges. *IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*. (pp. 46-57). https://doi.org/10.1109/MSR52588.2021.00018

Alamin, M. A. A., Uddin, G., Malakar, S., Afroz, S., Haider, T., & Iqbal, A. (2023). Developer discussion topics on the adoption and barriers of low code software development platforms. *Empirical Software Engineering*, 28(1), 1-59. https://doi.org/10.1007/s10664-022-10244-0

Bedini, F., Maschotta, R., & Zimmermann, A. (2021, April). A generative Approach for creating Eclipse Sirius Editors for generic Systems. In 2021 IEEE International Systems Conference (SysCon) (pp. 1-8). IEEE. https://doi.org/10.1109/SysCon48628.2021.9447062

Bhattacharyya, S. S., & Kumar, S. (2021). Study of deployment of "low code no code" applications toward improving digitization of supply chain management. *Journal of Science and Technology Policy Management*, 14(2), 271-278. https://doi.org/10.1108/JSTPM-06-2021-0084

Bock, A. C., & Frank, U. (2021). Low-code platform. *Business & Information Systems Engineering*, 63, 733-740. https://doi.org/10.1007/s12599-021-00726-8

Booch, G. (2018). The history of software engineering. *IEEE Software*, 35(5), 108-114. https://doi.org/10.1109/MS.2018.3571234

Boubeta-Puig, J., Rosa-Bilbao, J., & Mendling, J. (2021). CEPchain: A graphical model-driven solution for integrating complex event processing and blockchain. *Expert Systems with Applications*, *184*, 115578. https://doi.org/10.1016/j.eswa.2021.115578

Brambilla, M., Cabot, J., Wimmer, M. (2017). *Model-Driven Software Engineering in Practice*. Synthesis Lectures on Software Engineering,

2nd edition. Morgan & Claypool Publishers, San Rafael. https://doi.org/10.2200/S00751ED2V01Y201701SWE004

Brosch, P., Seidl, M., Wieland, K., Wimmer, M., & Langer, P. (2009). We can work it out: Collaborative conflict resolution in model versioning. *In ECSCW 2009* (pp. 207-214). Springer London. https://doi.org/10.1007/978-1-84882-854-4_12

Bruneliere, H., Cabot, J., Dupé, G., & Madiot, F. (2014). Modisco: A model driven reverse engineering framework. *Information and Software Technology*, 56(8), 1012-1032. https://doi.org/10.1016/j.infsof.2014.04.007

Bucaioni, A., Cicchetti, A., & Ciccozzi, F. (2022). Modelling in low-code development: a multi-vocal systematic review. *Software and Systems Modeling*, 21(5), 1959-1981. https://doi.org/10.1007/s10270-021-00964-0

Cabot, J. (2020). Positioning of the low-code movement within the field of model-driven engineering. In *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings* (pp. 1-3). https://doi.org/10.1145/3417990.3420210

ElBatanony, A., & Succi, G. (2021, October). Towards the no-code era: A vision and plan for the future of software development. In *Proceedings of the 1st ACM SIGPLAN International Workshop on Beyond Code: No Code* (pp. 29-35). https://doi.org/10.1145/3486949.3486965

Gottschalk, S., Bhat, R., Weidmann, N., Kirchhoff, J., & Engels, G. (2022). Low-code experimentation on software products. In *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings* (pp. 798-807). https://doi.org/10.1145/3550356.3561572

Guizzardi, G., & Proper, H. A. (2021). On understanding the value of domain modeling. In *Proceedings of 15th International Workshop on Value Modelling and Business Ontologies (VMBO 2021).* (pp. 51-62). https://ceur-ws.org/Vol-2835/paper6.pdf

Gürcan, F., Taentzer, G. (2021). Using Microsoft PowerApps, Mendix and OutSystems in two development scenarios: an experience report. *In 2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)* (pp. 67-72). IEEE. https://doi.org/10.1109/MODELS-C53483.2021.00017

Harkes, D. C. (2019). *Declarative Specification of Information System Data Models and Business Logic* (Doctoral dissertation, Delft University of Technology, Netherlands). https://doi.org/10.4233/uuid:5e9805ca-95d0-451e-a8f0-55decb26c94a

Hettel, T., Lawley, M., Raymond, K. (2008). Model synchronisation: Definitions for round-trip engineering. *International Conference on Theory and Practice of Model Transformations. Springer, Berlin, Heidelberg*. (pp. 31-45). https://doi.org/10.1007/978-3-540-69927-9_3

Hey, A., Hey, T., Pápay, G. (2014). *The computing universe: a journey through a revolution*, Cambridge University Press.

Hili, N., & Oliveira, R. A. (2022). A light-weight low-code platform for back-end automation. In *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings* (pp. 837-846). https://doi.org/10.1145/3550356.3561590

Käss, S., Strahringer, S., & Westner, M. (2023). Practitioners' Perceptions on the Adoption of Low Code Development Platforms. IEEE Access, 11, 29009-29034. https://doi.org/10.1109/ACCESS.2023.3258539

Kenneweg, B., Kasam, I., & McMullen, M. (2021). *Building Low-Code Applications with Mendix: Discover Best Practices and Expert* Techniques to Simplify Enterprise Web Development (pp. 1525-1551). Birmingham: Packt Publishing.

Khorram, F., Mottu, J. M., & Sunyé, G. (2020). Challenges & opportunities in low-code testing. In *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings* (pp. 1-10). https://doi.org/10.1145/3417990.3420204

Kolovos, D. S., Paige, R. F., & Polack, F. A. (2006). Eclipse development tools for epsilon. In *Eclipse summit Europe, eclipse modeling symposium* (Vol. 20062, p. 200). https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=8268b0795e2af075b4d0f78d1cc558dd69b24ea7

Kung, D. (2013). Object-oriented Software Engineering. In *An Agile Unified Methodology*. McGraw-Hill Higher Education.

Martins, R., Caldeira, F., Sa, F., Abbasi, M., & Martins, P. (2020). An overview on how to develop a low-code application using OutSystems. In *2020 International Conference on Smart Technologies in Computing, Electrical and Electronics (ICSTCEE)* (pp. 395-401). IEEE. https://doi.org/10.1109/ICSTCEE49637.2020.9277404

Mayerhofer, T., Langer, P., Wimmer, M., Kappel, G. (2013). XMOF: Executable DSMLs based on fUML. *International conference on software language engineering*. (pp. 56-75). https://doi.org/10.1007/978-3-319-02654-1_4

Mernik, M., Heering, J., & Sloane, A. M. (2005). When and how to develop domain-specific languages. *ACM computing surveys (CSUR)*, 37(4), 316-344. https://doi.org/10.1145/1118890.1118892

Nekrasov, A. (2022). *SwiftUI. In Swift Recipes for iOS Developers: Real-Life Code from App Store Apps* (pp. 319-341). Berkeley, CA: Apress. https://doi.org/10.1007/978-1-4842-8098-0_8

Overeem, M. (2022). *Evolution of Low-Code Platforms* (Doctoral dissertation, Utrecht University). https://dspace.library.uu.nl/handle/1874/420601

Rekstad, K. (2021). *A Modeling Environment in the Cloud for Education*. Master's thesis, NTNU. https://ntnuopen.ntnu.no/ntnu-xmlui/bitstream/handle/11250/2837177/no.ntnu:inspera:74730513:26437651.pdf?sequence=1

Richardson, C., Rymer, J. R. (2014). *New Development Platforms Emerge For Customer-Facing Applications*. Forrester: Cambridge, MA, USA. https://www.forrester.com/report/New-Development-Platforms-Emerge-For-CustomerFacing-Applications/RES113411

Rokis, K., & Kirikova, M. (2022, September). Challenges of Low-Code/No-Code Software Development: A Literature Review. *In 21st International Conference on Business Informatics Research, BIR.* (pp. 3-17). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-031-16947-2_1

Ruscio, D., Kolovos, D., Lara, J., Pierantonio, A., Tisi, M., Wimmer, M. (2022). Low-code development and model-driven engineering: Two sides of the same coin? *Journal of Software and Systems Modeling*, 1-10. https://doi.org/10.1007/s10270-021-00970-2

Sahay, A., Indamutsa, A., Ruscio, D., Pierantonio, A. (2020). Supporting the understanding and comparison of low-code development platforms. *Proceedings of 46th Euromicro Conference on Software Engineering and Advanced Applications SEAA*. (pp. 171–178). https://doi.org/10.1109/SEAA51224.2020.00036

Samimi-Dehkordi, L., Zamani, B., & Kolahdouz-Rahimi, S. (2019). Leveraging product line engineering for the development of domain-specific metamodeling languages. Journal of Computer Languages, 51, 193-213. https://doi.org/10.1016/j.cola.2019.02.006

Tisi, M., Mottu, J., Kolovos, D. S., Lara, J., Guerra, E., Ruscio, D., Pierantonio, A., and Wimmer, M. (2019). Lowcomote: Training the next generation of experts in scalable low-code engineering platforms. In *STAF (Co-Located Events)*. (pp. 73-78). https://ceur-ws.org/Vol-2405/13_paper.pdf

Tolvanen, J. P., Kelly, S. (2008). *Domain-specific modeling: Enabling full code generation*. Wiley IEEE Computer Society. http://eu.wiley.com/WileyCDA/WileyTitle/productCd-0470036664.html

Verbruggen, C., & Snoeck, M. (2023). Practitioners' experiences with

model-driven engineering: a meta-review. *Software and Systems Modeling*, 22(1), 111-129. https://doi.org/10.1007/s10270-022-01020-1

Vincent, P., Iijima, K., Driver, M., Wong, J., & Natis, Y. (2019). *Magic quadrant for enterprise low-code application platforms*. Gartner report. https://www.zoho.com/sites/zweb/images/zoho_general_pages/magic-quadrant-for-enterprise-low-code-shareable-until-end-of-year-only.pdf

Wei, Y., & Blake, M. B. (2010). Service-oriented computing and cloud computing: Challenges and opportunities. IEEE Internet Computing, 14(6), 72-75. https://doi.org/10.1109/MIC.2010.147

Wurster, M., Breitenbücher, U., Falkenthal, M., Krieger, C., Leymann, F., Saatkamp, K., & Soldani, J. (2020). The essential deployment metamodel: a systematic review of deployment automation technologies. *SICS Software-Intensive Cyber-Physical Systems*, *35*, 63-75. https://doi.org/10.1007/s00450-019-00412-x

Zhao, J. F., & Zhou, J. T. (2014). Strategies and methods for cloud migration. International Journal of Automation and Computing, 11(2), 143-152. https://doi.org/10.1007/s11633-014-0776-7

Zheng, Z., Xie, S., Dai, H. N., Chen, W., Chen, X., Weng, J., & Imran, M. (2020). An overview on smart contracts: Challenges, advances and platforms. *Future Generation Computer Systems*, 105, 475-491. https://doi.org/10.1016/j.future.2019.12.019

References [In Persian]

Shahsavari-pour, N., Rezvandoust, S., Mirzaie, A., Heydarbeig, S. (2017). An Investigation into the Relationship between Alignment of IT Strategy and Business Strategy with Enterprise Agility in Software Companies. *Business Intelligence Management Studie*, 5(19), 75-103 https://doi.org/10.22054/ims.2017.7055 [In Persian]

استناد به این مقاله: صمیمی دهکردی، لیلا.، حری، عباس. (۱۴۰۲). نقش متقابل مهندسی نرم‌افزار مدل‌رانده و سکوی توسعه کم‌کد، *مطالعات مدیریت کسب وکار هوشمند*، ۱۱(۴۴)، ۲۲۵-۲۵۷.

DOI: 10.22054/ims.2023.70563.2245