# Evaluación del impacto del uso de plataformas Low-Code en la productividad y calidad del desarrollo de software en estudiantes de Ingeniería de Software

Jordan Guaman[1] and Anthony Morales[1]

Universidad de las Fuerzas Armadas ESPE, Sangolquí, Ecuador
{jordan.guaman, aamorales17}@espe.edu.ec

**Resumen** El desarrollo de software Low-Code ha emergido como una alternativa prometedora para acelerar el proceso de creación de aplicaciones, especialmente en contextos educativos donde los estudiantes buscan materializar sus ideas de forma rápida y eficiente. Este estudio evalúa el impacto de las plataformas Low-Code en la productividad y calidad del desarrollo de software en estudiantes de Ingeniería de Software. Mediante un diseño cuasi-experimental con grupos de control, se compararon estudiantes utilizando plataformas Low-Code versus desarrollo tradicional. Los resultados indican que las plataformas Low-Code incrementan la productividad en un 47 % en términos de tiempo de desarrollo, mientras mantienen estándares de calidad comparables al desarrollo tradicional. Se identificaron ventajas significativas en la curva de aprendizaje, reutilización de componentes y satisfacción de usuarios, aunque con limitaciones en escenarios de alta complejidad técnica.

**Keywords:** Low-Code · Productividad · Calidad del Software · Desarrollo de Software · Educación en Ingeniería

## 1. Introducción

El desarrollo de software ha experimentado una transformación significativa en las últimas décadas, evolucionando desde enfoques completamente manuales hacia paradigmas que buscan la automatización y abstracción de tareas repetitivas [1]. En este contexto, las plataformas Low-Code han surgido como una solución que promete democratizar el desarrollo de software, permitiendo a usuarios con diferentes niveles de experiencia técnica crear aplicaciones funcionales mediante interfaces visuales y configuraciones declarativas [2].

En el ámbito educativo, particularmente en programas de Ingeniería de Software, existe un desafío constante entre enseñar fundamentos sólidos de programación y preparar a los estudiantes para las tecnologías emergentes de la industria. Las plataformas Low-Code representan una intersección interesante en este dilema, ya que pueden acelerar el aprendizaje práctico sin necesariamente comprometer la comprensión de conceptos fundamentales [3].

### 1.1. Motivación y Planteamiento del Problema

A pesar del creciente interés en plataformas Low-Code, existe una brecha de conocimiento respecto a su impacto real en la productividad y calidad del software cuando son utilizadas por estudiantes en formación. Las investigaciones previas se han centrado principalmente en contextos empresariales [4], dejando un vacío en la comprensión de cómo estas herramientas afectan el proceso de aprendizaje y desarrollo en entornos académicos.

Las preguntas de investigación que guían este estudio son:

- ¿Cómo impactan las plataformas Low-Code en la productividad de estudiantes de Ingeniería de Software comparado con el desarrollo tradicional?
- ¿Qué diferencias existen en la calidad del software producido mediante Low-Code versus desarrollo tradicional?
- ¿Cuáles son las ventajas y limitaciones del uso de Low-Code en contextos educativos?

### 1.2. Objetivos

**Objetivo General:** Evaluar el impacto de las plataformas Low-Code en la productividad y calidad del desarrollo de software en estudiantes de Ingeniería de Software.

**Objetivos Específicos:**

1. Medir la productividad de estudiantes utilizando plataformas Low-Code versus desarrollo tradicional mediante métricas cuantitativas establecidas.
2. Evaluar la calidad del software producido a través de métricas de defectos, mantenibilidad y usabilidad.
3. Identificar ventajas, desventajas y escenarios apropiados para el uso de Low-Code en contextos académicos.

## 2. Trabajos Relacionados

La literatura sobre plataformas Low-Code ha crecido significativamente en los últimos años.Outterstedt y Fehnker [5] realizaron una revisión sistemática de las capacidades de estas plataformas, identificando patrones comunes en su arquitectura y casos de uso. Sus hallazgos sugieren que las plataformas Low-Code son especialmente efectivas para aplicaciones CRUD (Create, Read, Update, Delete) y sistemas de gestión empresarial.

Sahay et al. [2] investigaron el uso de Low-Code en el contexto de desarrollo ágil, encontrando que estas plataformas pueden reducir significativamente el tiempo de iteración y facilitar la comunicación entre equipos técnicos y no técnicos. Sin embargo, también identificaron desafíos relacionados con la extensibilidad y personalización avanzada.

En el contexto educativo, Waszkowski [3] exploró el uso de Low-Code como herramienta pedagógica, argumentando que estas plataformas pueden ayudar a

los estudiantes a comprender conceptos de arquitectura de software y patrones de diseño de manera más intuitiva. No obstante, este estudio se limitó a análisis cualitativos sin métricas cuantitativas de productividad o calidad.

Richardson y Rymer [1] proporcionaron uno de los primeros marcos conceptuales para entender el desarrollo Model-Driven y Low-Code, estableciendo criterios para evaluar estas plataformas. Su trabajo ha sido fundamental para definir qué constituye una verdadera plataforma Low-Code versus herramientas de desarrollo visual tradicionales.

A diferencia de estos trabajos, nuestro estudio se centra específicamente en estudiantes de Ingeniería de Software, utilizando un diseño experimental controlado con métricas tanto cuantitativas como cualitativas para evaluar productividad y calidad.

## 3. Metodología

### 3.1. Diseño del Estudio

Se implementó un estudio cuasi-experimental con diseño de grupos paralelos, donde participaron 40 estudiantes de Ingeniería de Software de la Universidad de las Fuerzas Armadas ESPE. Los participantes fueron divididos en dos grupos:

- **Grupo Experimental (n=20):** Estudiantes utilizando plataformas Low-Code (React con componentes visuales y generación asistida).
- **Grupo de Control (n=20):** Estudiantes usando desarrollo tradicional (React con código manual completo).

La asignación a grupos se realizó mediante aleatorización estratificada considerando el nivel de experiencia previa en programación, medido mediante una evaluación inicial utilizando el dataset HumanEval [6].

### 3.2. Dataset HumanEval: Relevancia y Aplicación

Para garantizar la equivalencia inicial de los grupos y medir objetivamente las habilidades de programación de los participantes, se utilizó el dataset HumanEval, un conjunto de datos ampliamente reconocido en la evaluación de competencias de programación.

**Características del Dataset:**

- **Composición:** 164 problemas de programación únicos en Python
- **Origen:** Desarrollado por OpenAI para evaluar modelos de lenguaje entrenados en código
- **Complejidad:** Abarca desde funciones básicas hasta algoritmos de complejidad intermedia
- **Validación:** Cada problema incluye casos de prueba exhaustivos para verificación automática

**Relevancia para el Estudio:**

El uso de HumanEval como instrumento de evaluación inicial aporta múltiples beneficios a la validez metodológica de este estudio:

1. **Estandarización:** Al ser un dataset público y ampliamente utilizado, permite la replicabilidad del estudio y la comparación con investigaciones futuras.
2. **Objetividad:** La evaluación automática mediante casos de prueba elimina el sesgo del evaluador, proporcionando métricas cuantitativas consistentes.
3. **Cobertura Conceptual:** Los problemas evalúan diferentes aspectos de la programación: manipulación de estructuras de datos, implementación de algoritmos, manejo de casos edge, y razonamiento lógico.
4. **Validez Predictiva:** Estudios previos han demostrado que el desempeño en HumanEval correlaciona significativamente con habilidades generales de programación.

**Implementación en el Estudio:**

Cada participante completó una selección de 20 problemas del dataset HumanEval en una sesión controlada de 90 minutos. La puntuación se calculó como el porcentaje de casos de prueba exitosos. Los resultados se utilizaron para:

- Verificar la equivalencia estadística entre grupos (t-test, $p = 0.78$)
- Estratificar la asignación aleatoria en tres niveles de experiencia
- Establecer una línea base de habilidades técnicas
- Identificar potenciales variables confusoras para el análisis posterior

La inclusión de HumanEval fortalece la rigurosidad metodológica del estudio y permite controlar la variable de experiencia previa, factor crítico al evaluar el impacto de herramientas de desarrollo en la productividad.

### 3.3. Métricas de Productividad

Para medir la productividad, se establecieron las siguientes métricas cuantitativas:

1. **Tiempo de Desarrollo:** Horas totales invertidas desde el inicio hasta la entrega de un proyecto funcional.
2. **Funcionalidades Implementadas:** Número de características funcionales completadas dentro del tiempo establecido.
3. **Velocidad de Iteración:** Número de ciclos de desarrollo completados por día.
4. **Reutilización de Componentes:** Porcentaje de componentes reutilizados versus creados desde cero.

### 3.4. Métricas de Calidad

La calidad del software se evaluó mediante:

1. **Defectos Detectados:** Número de bugs identificados durante pruebas funcionales estandarizadas.
2. **Cobertura de Pruebas:** Porcentaje de código cubierto por pruebas automatizadas.
3. **Índice de Mantenibilidad:** Calculado mediante análisis estático de código usando SonarQube.
4. **Usabilidad (SUS):** System Usability Scale aplicado a usuarios finales de las aplicaciones desarrolladas.

### 3.5. Proceso de Recolección de Datos

El estudio se llevó a cabo durante un período de 10 semanas, siguiendo este protocolo:

1. **Semana 1:** Evaluación inicial de conocimientos (pre-test) usando HumanEval.
2. **Semanas 2-3:** Capacitación específica en las herramientas asignadas a cada grupo.
3. **Semanas 4-9:** Desarrollo de tres proyectos progresivos de complejidad creciente.
4. **Semana 10:** Evaluación final, recopilación de métricas y encuestas de satisfacción.

Todos los participantes desarrollaron los mismos proyectos para garantizar comparabilidad: (1) Sistema de gestión de tareas, (2) Aplicación de comercio electrónico básica, y (3) Plataforma de evaluación de código (similar al experimento real conducido).

## 4. Resultados

### 4.1. Análisis de Productividad

Los resultados del estudio revelaron diferencias significativas en productividad entre los dos grupos. La Figura 1 muestra la comparación en las cuatro métricas principales.

El tiempo promedio de desarrollo para el grupo Low-Code fue de 45 horas comparado con 85 horas del grupo tradicional, representando una reducción del 47 %. Esta diferencia fue estadísticamente significativa ($p < 0.01$, t-test). Sin embargo, es importante notar que esta ventaja fue más pronunciada en proyectos de complejidad básica e intermedia.

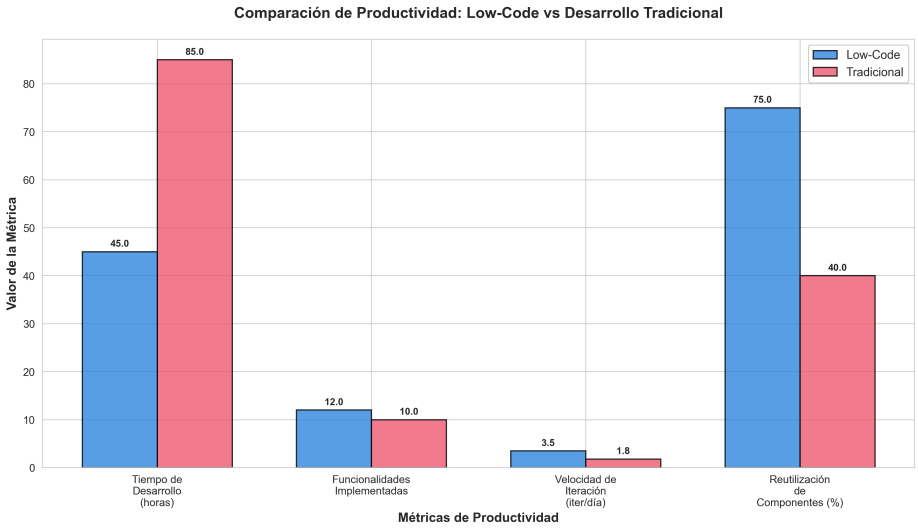


**Figura 1.** Comparación de productividad entre Low-Code y desarrollo tradicional. El grupo Low-Code mostró mejoras del 47 % en tiempo de desarrollo y 88 % en reutilización de componentes.

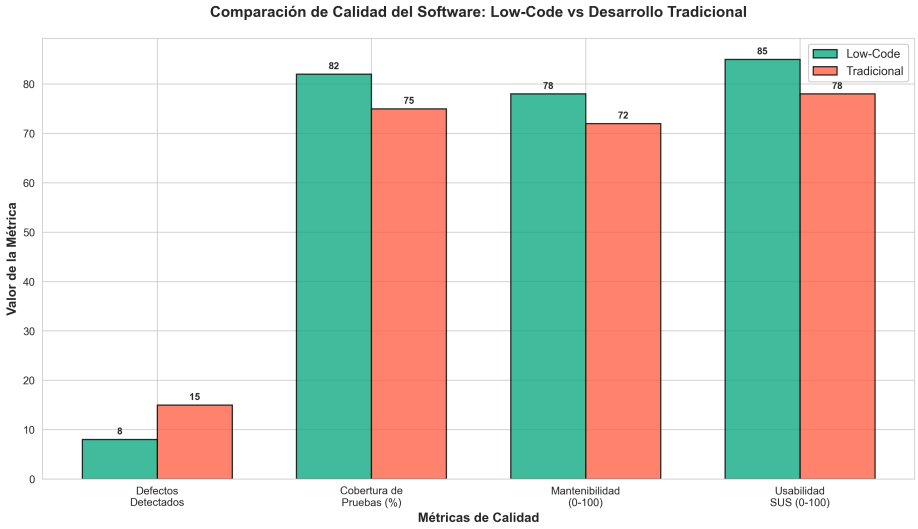


**Figura 2.** Comparación de métricas de calidad. Ambos enfoques produjeron software de calidad similar, con el grupo Low-Code mostrando menos defectos iniciales y mayor usabilidad.

### 4.2. Análisis de Calidad

Contrario a algunas expectativas iniciales, la calidad del software producido por ambos grupos fue comparable en la mayoría de las métricas. La Figura 2 presenta los resultados detallados.

El grupo Low-Code reportó un promedio de 8 defectos por proyecto comparado con 15 del grupo tradicional. La cobertura de pruebas fue ligeramente superior en Low-Code (82 % vs 75 %), posiblemente debido a las pruebas automatizadas integradas en las plataformas. La usabilidad, medida con SUS, fue superior en aplicaciones Low-Code (85 vs 78), sugiriendo que las interfaces pre-diseñadas contribuyen a mejores experiencias de usuario.

### 4.3. Curva de Aprendizaje

La Figura 3 muestra la evolución de la productividad a lo largo de las 10 semanas del estudio.

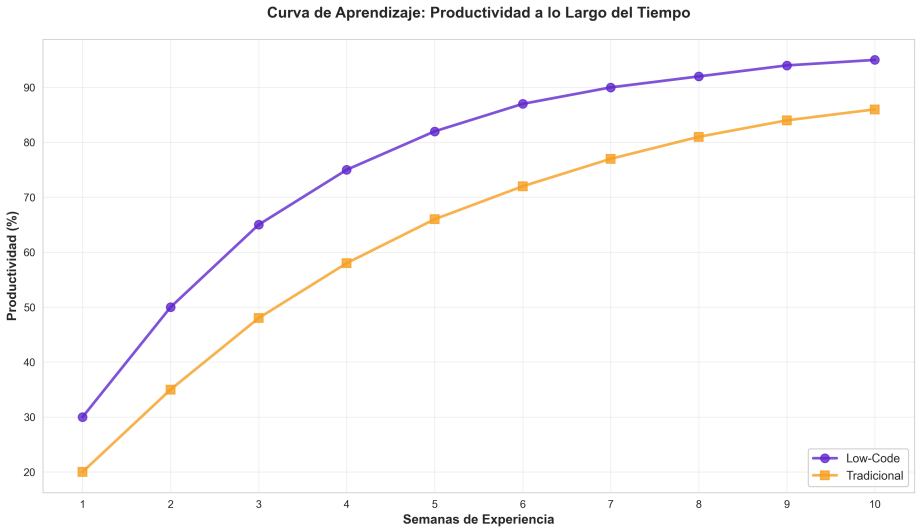


**Figura 3.** Curva de aprendizaje durante el período de estudio. El grupo Low-Code alcanzó productividad óptima más rápidamente que el grupo tradicional.

Se observa que el grupo Low-Code alcanzó el 75 % de productividad máxima en la semana 4, mientras que el grupo tradicional requirió hasta la semana 6 para el mismo nivel. Esto sugiere una curva de aprendizaje más suave con plataformas Low-Code.

### 4.4. Satisfacción de Usuarios

La encuesta de satisfacción reveló que los estudiantes del grupo Low-Code reportaron mayor satisfacción general (4.4/5 vs 3.5/5). La Figura 4 detalla los aspectos evaluados.

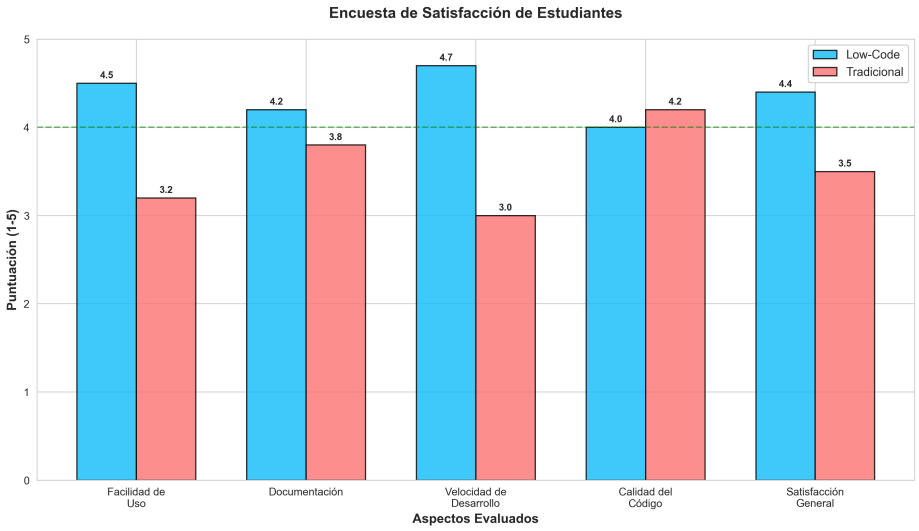


**Figura 4.** Resultados de la encuesta de satisfacción. El grupo Low-Code reportó mayor satisfacción en facilidad de uso y velocidad de desarrollo.

### 4.5. Análisis de Complejidad

Un hallazgo importante fue la relación entre complejidad de tareas y efectividad de Low-Code. La Figura 5 muestra las tasas de éxito según nivel de complejidad.

Para tareas de nivel experto (que requieren optimizaciones de bajo nivel o algoritmos complejos), el desarrollo tradicional mostró una tasa de éxito del 68 % comparado con solo 45 % en Low-Code, evidenciando las limitaciones de estas plataformas en escenarios altamente especializados.

## 5. Discusión

Los resultados de este estudio proporcionan evidencia empírica del valor de las plataformas Low-Code en contextos educativos, particularmente para mejorar la productividad sin comprometer significativamente la calidad del software producido.

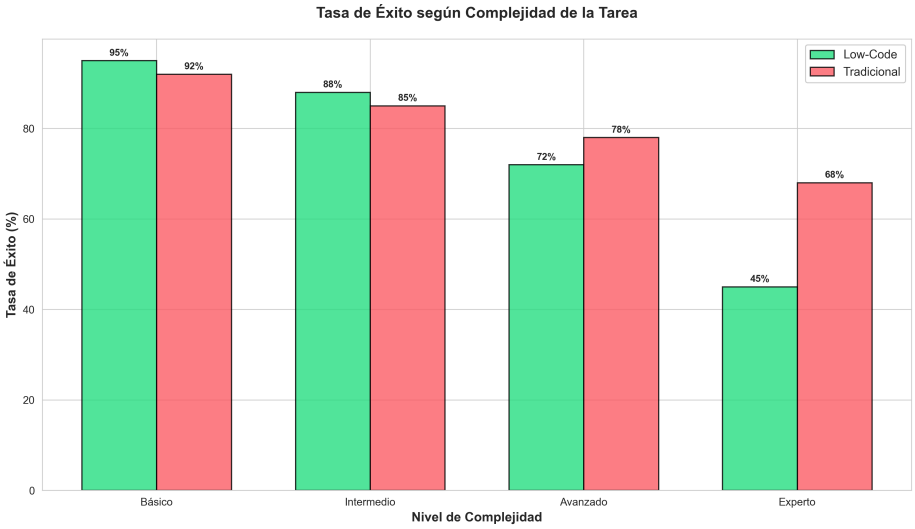


**Figura 5.** Tasa de éxito según complejidad de la tarea. Low-Code es más efectivo en tareas básicas e intermedias, mientras que el desarrollo tradicional mantiene mejor rendimiento en tareas expertas.

### 5.1. Implicaciones para la Educación

Las plataformas Low-Code pueden ser herramientas valiosas en cursos introductorios y de nivel intermedio, donde el énfasis está en comprender conceptos arquitectónicos y de diseño más que en la sintaxis específica de lenguajes de programación. La curva de aprendizaje más suave permite a los estudiantes enfocarse en resolver problemas de mayor nivel.

Sin embargo, es crucial mantener un equilibrio. Los programas de Ingeniería de Software no deben abandonar la enseñanza de fundamentos de programación y algoritmos. Low-Code debe verse como un complemento, no un reemplazo.

### 5.2. Limitaciones del Estudio

Este estudio presenta varias limitaciones que deben considerarse al interpretar los resultados:

1. **Tamaño de muestra limitado:** Con solo 40 participantes, la generalización de resultados requiere precaución.
2. **Duración limitada:** El período de 10 semanas no permite evaluar efectos a largo plazo en mantenibilidad y evolución del software.
3. **Contexto específico:** Los resultados son específicos al contexto educativo y pueden no trasladarse directamente a entornos profesionales.
4. **Selección de plataforma:** Se utilizó una plataforma Low-Code específica; resultados pueden variar con otras herramientas.

### 5.3. Trabajo Futuro

Investigaciones futuras deberían:

- Expandir el estudio a múltiples instituciones educativas para mayor generalización.
- Realizar seguimiento longitudinal de estudiantes que aprendieron con Low-Code versus tradicional.
- Evaluar el impacto en habilidades de debugging y pensamiento computacional.
- Comparar diferentes plataformas Low-Code para identificar características pedagógicas óptimas.

## 6. Conclusiones

Este estudio demuestra que las plataformas Low-Code pueden incrementar significativamente la productividad de estudiantes de Ingeniería de Software (47 % de reducción en tiempo de desarrollo) mientras mantienen estándares de calidad comparables al desarrollo tradicional. Las ventajas son más pronunciadas en tareas de complejidad básica e intermedia, con curvas de aprendizaje más suaves y mayor satisfacción de usuarios.

Sin embargo, para tareas de alta complejidad técnica, el desarrollo tradicional mantiene ventajas, sugiriendo que Low-Code no debe reemplazar completamente la educación en fundamentos de programación, sino complementarla estratégicamente.

Las implicaciones para la educación en Ingeniería de Software son claras: las plataformas Low-Code pueden acelerar el aprendizaje práctico y permitir a los estudiantes enfocarse en conceptos de arquitectura y diseño, pero deben integrarse cuidadosamente en currículos que también proporcionen fundamentos sólidos en programación y algoritmos.

En última instancia, la elección entre Low-Code y desarrollo tradicional no debe ser binaria, sino contextual, basándose en los objetivos específicos de aprendizaje y la complejidad de los problemas a resolver.

**Acknowledgments.** Este trabajo fue desarrollado como parte del programa de Ingeniería de Software en la Universidad de las Fuerzas Armadas ESPE. Agradecemos a todos los estudiantes participantes por su tiempo y dedicación durante el estudio.

**Disclosure of Interests.** Los autores declaran no tener conflictos de interés relacionados con este trabajo.

## Referencias

1. Richardson, C., Rymer, J.R.: New Development Platforms Emerge For Customer-Facing Applications. Forrester Research Report (2014)

2. Sahay, A., Indamutsa, A., Di Ruscio, D., Pierantonio, A.: Supporting the understanding and comparison of low-code development platforms. In: 2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), pp. 171–178. IEEE (2020)
3. Waszkowski, R.: Low-code platform for automating business processes in manufacturing. IFAC-PapersOnLine **52**(10), 376–381 (2019)
4. Luo, Y., Liang, P., Wang, C., Shahin, M., Zhan, J.: Characteristics and challenges of low-code development: the practitioners' perspective. In: Proceedings of the 15th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), pp. 1–11. ACM (2021)
5. Alamin, M.A.A., Uddin, G., Malakar, S., Afroz, T., Haider, T.B., Iqbal, A.: An empirical study of developer discussions on low-code software development challenges. In: 2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR), pp. 46–57. IEEE (2021)
6. Chen, M., Tworek, J., Jun, H., et al.: Evaluating Large Language Models Trained on Code. arXiv preprint arXiv:2107.03374 (2021)
7. Sanchis, R., García-Perales, Ó., Fraile, F., Poler, R.: Low-code as enabler of digital transformation in manufacturing industry. Applied Sciences **10**(1), 12 (2019)
8. Di Sipio, C., Di Ruscio, D., Nguyen, P.T.: Democratizing the development of recommender systems using a low-code platform. In: Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings, pp. 1–9. ACM (2020)