Low-Code Development and EUP Learning Barriers

Thiago Rocha Silva (trsi@mmmi.sdu.dk)
Associate Professor

EUD - Fall 2024

What’s happening now?

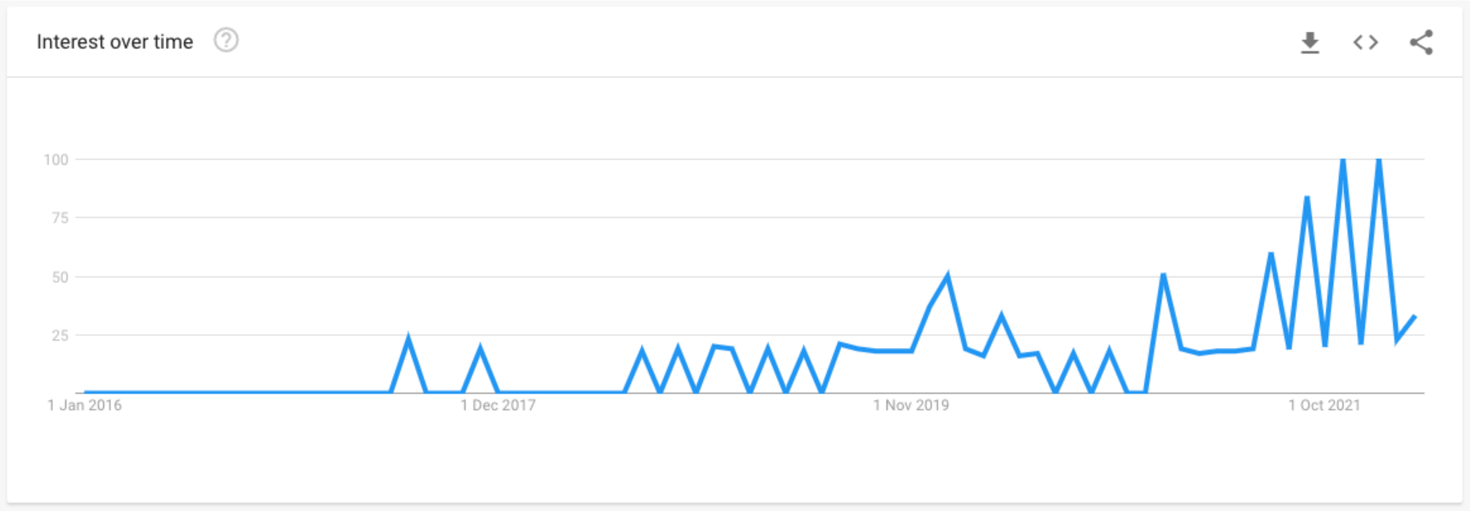
- While the demand for software systems is **exploding**, there is a severe **shortage of software developers** to cope with this demand, and it **won’t become any better** in the near future[1].
- There is a **digital native workforce**:
  - ❖ **Computer literacy** has **improved dramatically** over the last years.
  - ❖ The **basics of computer programming** are now taught in many countries as part of **compulsory education**.
- **Cloud-based services** became a reality (SaaS).
- **Training** is nowadays widely available on YouTube and other online media.
- **Collaborative development** is now supported by a wide range of tools (CSCW).
- Hiring **professional devs** has become **more and more expensive**, especially for SMEs.

[1] Breaux, T., Moritz, J. (2021). The 2021 software developer shortage is coming. Commun. ACM 64(7), 39–41.

What if “citizen developers”[1] could program the software they need?

[1] *The term refers to ordinary citizens, subject-matter experts, etc. who* ***do not know how to program***

Low-Code Development: Google Trends



Low-Code Development

Forbes

Jul 20, 2017, 01:20pm EDT

The Low-Code/No-Code Movement: More Disruptive Than You Realize

Jason Bloomberg Former Contributor

Enterpr

I write a

Subject Matter First

A Manifesto

Subject matter experts, or SMEs, own the knowledge and expertise that is the backbone of software and the foundation of digitalization. But too often this rich expertise is not captured in a structured way and gets lost when translating it for software developers who then analyze, interpret and understand it before writing code. With the rate of change increasing, time-to-market shortening and product variability blooming, this approach is increasingly untenable. It causes delays, quality problems and frustration for everybody involved. We advocate for adopting a mindset that puts subject SMEs directly in control of "their" part of the software and lets developers focus on their core skill, software engineering. Here is how we achieve it:

Empower subject matter experts to capture, understand, reason about data, structures, rules, behaviors and other forms of domain knowledge in a precise and unambiguous form	**by** providing them with tailored software tools that allow them to directly edit, validate, simulate and test that knowledge	**rather than** expecting experts to juggle complex subject matter in their minds, Word documents, user stories and other generic, non-optimized tools.

MSCA

Marie Skłodowska-Curie Actions

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement n° 813884.

lowcomote

Low-code development platforms allow non-programmers to build full applications by interacting through dynamic graphical user interfaces, visual diagrams and declarative languages.

the potential to

er the different

Springer

Call for Papers

Software and Systems Modeling

Theme Section:

Modeling in Low-Code Development Platforms

IMT Atlantique Bretagne-Pays de la Loire École Mines-Télécom

University of York

UAM

Università degli Studi dell'Aquila

JKU Johannes Kepler Universität Linz

BT

intecs Solutions

UGROUND

CLMS

IncQuery Labs

SPARX SYSTEMS

METADEV

The Open Group

aws

SDU

sdu.dk

#sdudk

Low-Code Development: Projections

- According to Gartner[1], by 2024, low-code application development will be responsible for **more than 65%** of the application development activity.
- Currently, estimates[2] point out that the low-code app development market is roughly around **$10 billion** globally.
- Microsoft PowerApps sees at least **7.4 million new builds** on its platform **every single month**.

[1] Vincent, P., Iijima, K., Driver, M., Jason, W., Natis, Y. (2016). Magic Quadrant for Enterprise Low-Code Application Platforms. Gartner.
[2] https://www.infoworld.com/article/3694173/why-generative-ai-will-turbocharge-low-code-and-no-code-development.html

Low-Code Development Platforms

K2
kissflow
TRACKVIA
appian
PRONTOFORMS
ninox
Betty Blocks
outsystems
DWKIT
aura quantic
NEWGEN
kintone
MICROPACT
amazon Honeycode BETA
VINYL
quixy
AppSheet
Bonitasoft
Retool
GeneXus SOFTWARE THAT MAKES SOFTWARE
agilepoint
Creatio
PEGA
Claris
salesforce
mx mendix
Microsoft PowerApps
servicenow
ZOHO
SwiftUI
LANSA An Idera, Inc. Company
MATS Easier done than said
unqork
nintex
wavemaker
caspio
bizagi
quickbase
ORACLE APEX
SDU

Low-Code Development Platforms (LCDPs)

- **Definition**:

*"Platforms that enable **rapid delivery** of **business applications** with a **minimum of hand-coding** and minimal upfront investment in setup, training, and deployment"*

Forrester (2014)

Richardson, C., Rymer, J.: New Development Platforms Emerge for Customer-Facing Applications. Forrester Research, Cambridge (2014).

Low-Code Development Platforms (LCDPs)

- **Definition**:

"Products and/or cloud services for ***application development*** *that employ* ***visual, declarative techniques instead of programming*** *and are available to customers at low- or no-cost in money and training time to begin, with costs rising in proportion of the business value of the platforms"*

Forrester (2017)

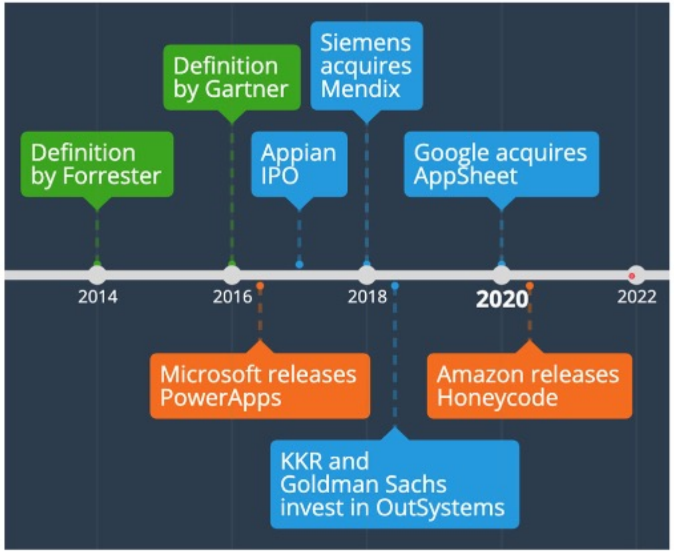
Rymer, J.: The Forrester Wave: Low-Code Development Platforms For AD&D Pros, Q4 2017. Forrester Research, Cambridge (2016)

Low-Code *vs* No-Code

- **No-code** development platform (NCDP) is a related term used for platforms that **eliminate the need for programming** using **visual languages**, **graphical user interfaces**, and **configuration**.
- The term is widely used in marketing, but **do not** clearly represent an specific market segment[1].

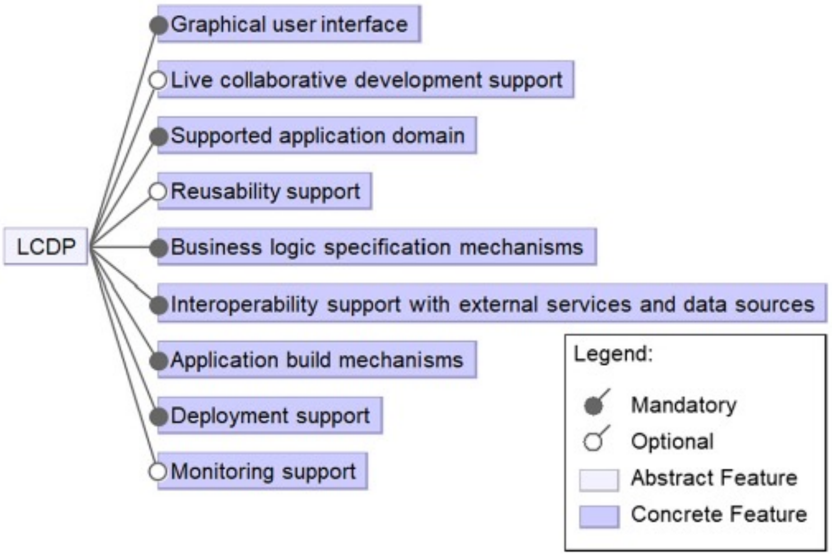
[1]Rymer, J., Koplowitz, R.: Now Tech: Rapid App Delivery, Q1 2019. Forrester Research, Cambridge (2019)

Low-Code Development Platforms (LCDPs)



Major events in low-code history (Di Ruscio et al., 2022).

Top-level features of LCDPs

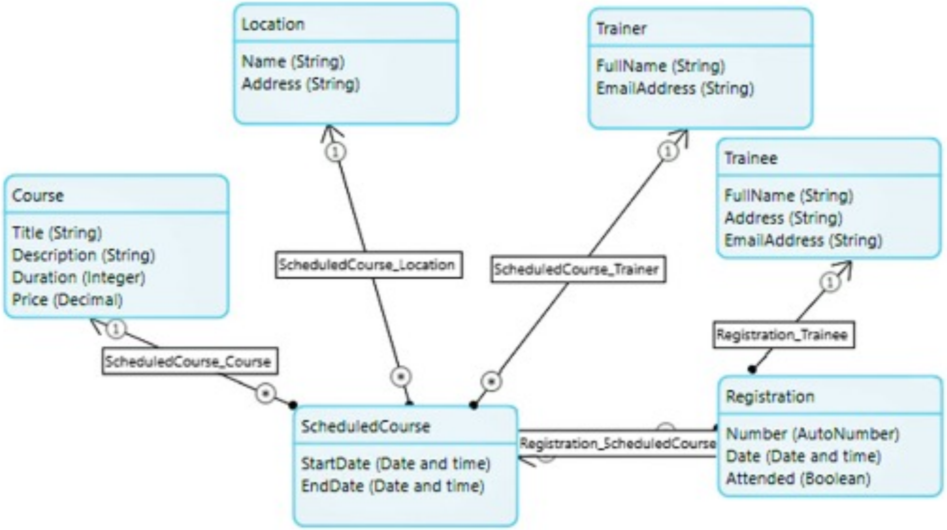


Source: Di Ruscio et al. (2022)

LCDPs: Tool-supported steps

- **Domain modelling**:

❖ Users are provided with modelling constructs to represent **concepts and relationships** underpinning the application being developed.

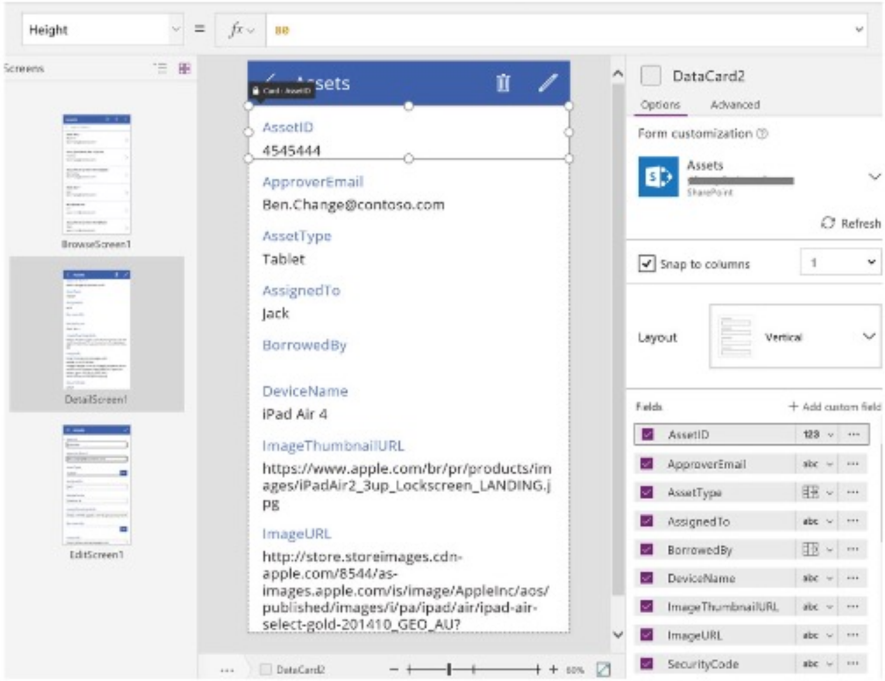


A simple domain model specified in **Mendix** (Sahay et al., 2020).

LCDPs: Tool-supported steps

- **User interface definition**:

❖ Users define **data forms and pages** to create, edit, and visualize data that the application under development will manage.

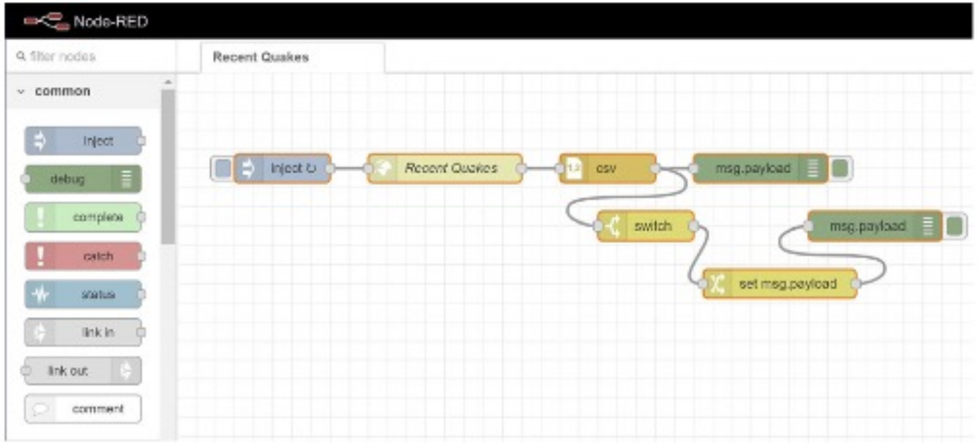


User interface definition with **Microsoft PowerApps**.

LCDPs: Tool-supported steps

- **Business logic specification**:

❖ Users define the **control and data flows** of the system under development through business logic specification mechanisms such as **graphical workflows** and **textual business rules**.

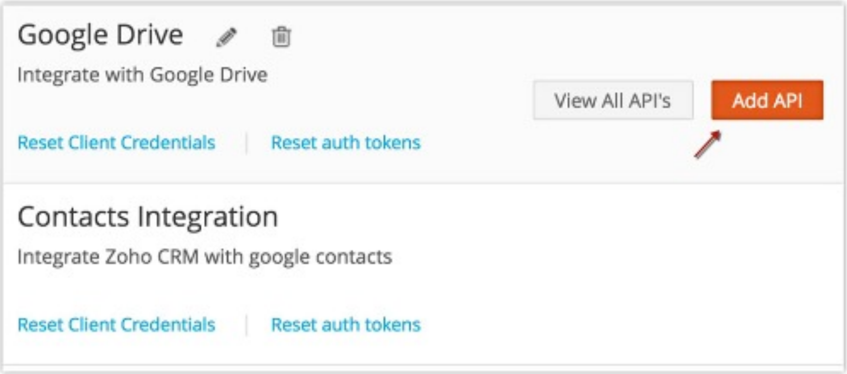


Business logic specification with **Node-RED**.

LCDPs: Tool-supported steps

- **Integration with external services**:

❖ LCDPs typically provide **interoperability support** with external services and data sources to **use services or consume data** provided by third-party systems, e.g., using dedicated APIs.

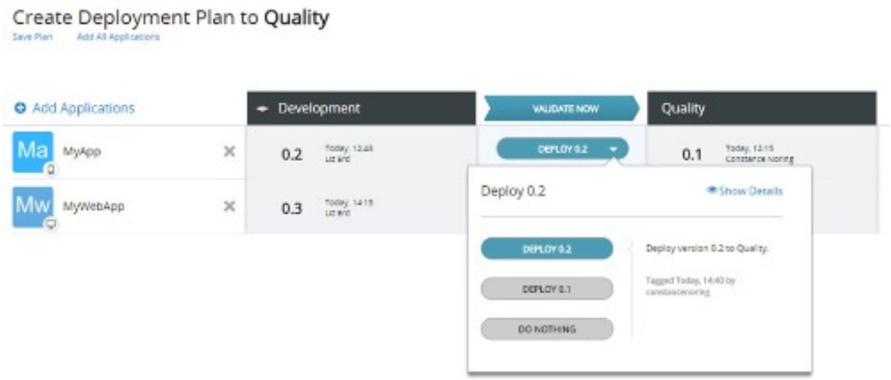


Configuring the Google Drive connector in **Zoho Creator**.

LCDPs: Tool-supported steps

- **Application generation and deployment**:

❖ LCDPs **generates and deploys** the modelled application by means of provided application build mechanisms. Deployments are typically done on **cloud infrastructures**.

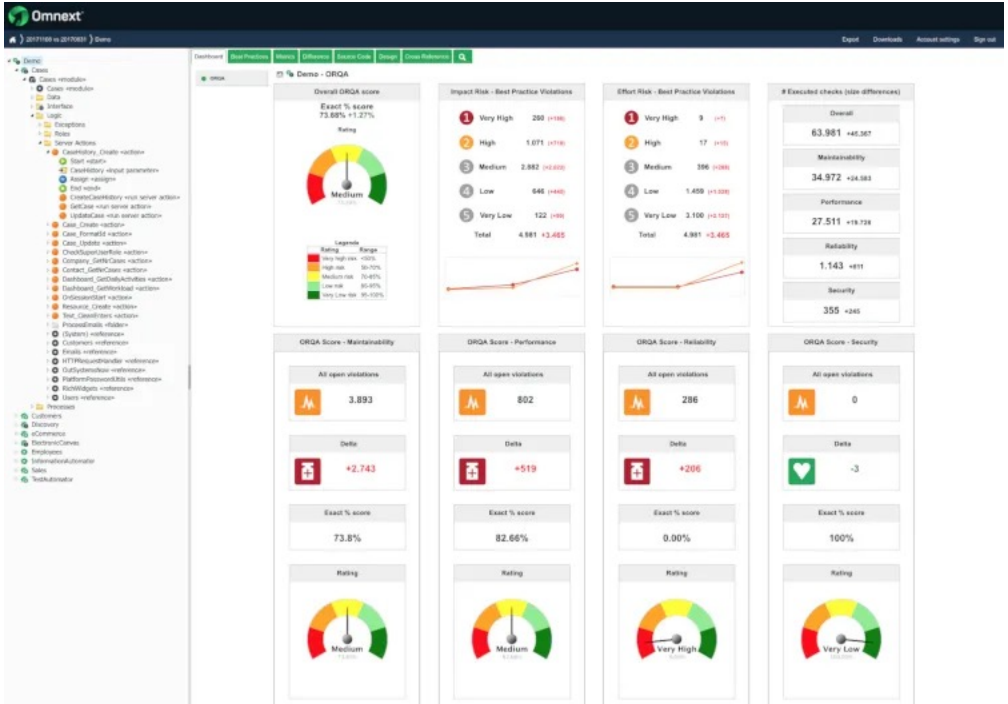


Application deployment with **OutSystems**.

LCDPs: Tool-supported steps

- **Application maintenance**:

❖ LCDPs provide mechanisms to **monitor and maintain** the developed system by means of dedicated features, e.g., to react in case of **unforeseen requirements** that need to be addressed or **fix issues** that might occur during the operation of the system.



Dashboard with **OutSystems**.

Typical features in LCDPs

Source: Sahay et al. (2020)

Feature	OutSystems	Mendix	Zoho Creator	MS PowerApp	Google App Maker	Kissflow	Salesforce App Cloud	Appian
Graphical user interface								
Drag-and-drop designer	✓	✓	✓		✓	✓	✓	✓
Point and click approach				✓				
Pre-built forms/reports	✓	✓	✓	✓	✓	✓	✓	✓
Pre-built dashboards	✓		✓	✓		✓	✓	
Forms			✓	✓				
Progress tracking	✓	✓	✓	✓	✓	✓	✓	✓
Advanced reporting						✓		
Built-in workflows			✓			✓	✓	
Configurable workflows			✓			✓	✓	
Interoperability support								
Interoperability with external service	✓	✓	✓	✓		✓	✓	✓
Connection with data sources	✓	✓	✓	✓	✓	✓	✓	✓
Security Support								
Application security	✓	✓	✓	✓	✓	✓	✓	✓
Platform security	✓	✓	✓	✓	✓	✓	✓	✓
Collaborative development support								
Off-line collaboration	✓	✓	✓	✓	✓	✓	✓	✓
On-line collaboration	✓	✓			✓	✓	✓	✓
Reusability support								
Built-in workflows			✓			✓	✓	
Pre-built forms/reports	✓	✓	✓	✓	✓	✓	✓	✓
Pre-built dashboards	✓		✓	✓		✓	✓	
Scalability								
Scalability on number of users	✓	✓	✓	✓	✓	✓	✓	✓
Scalability on data traffic	✓	✓	✓	✓	✓	✓	✓	
Scalability on data storage	✓	✓	✓	✓	✓	✓	✓	
Business logic specification mechanisms								
Business rules engine	✓	✓	✓	✓	✓	✓	✓	✓
Graphical workflow editor	✓	✓				✓	✓	
AI enabled business logic	✓					✓	✓	✓
Application build mechanisms								
Code generation	✓							
Models at run-time		✓	✓	✓	✓	✓	✓	✓
Deployment support								
Deployment on cloud	✓	✓	✓	✓	✓	✓	✓	✓
Deployment on local infrastructures	✓	✓					✓	✓
Kinds of supported applications								
Event monitoring	✓	✓	✓	✓	✓	✓	✓	✓
Process automation	✓		✓	✓	✓	✓		✓
Approval process control					✓			
Escalation management						✓		
Inventory management	✓	✓	✓	✓	✓	✓	✓	✓
Quality management		✓	✓	✓	✓	✓	✓	✓
Workflow management	✓	✓	✓	✓	✓	✓	✓	✓

Typical features in LCDPs

Source: Bock & Frank (2021)

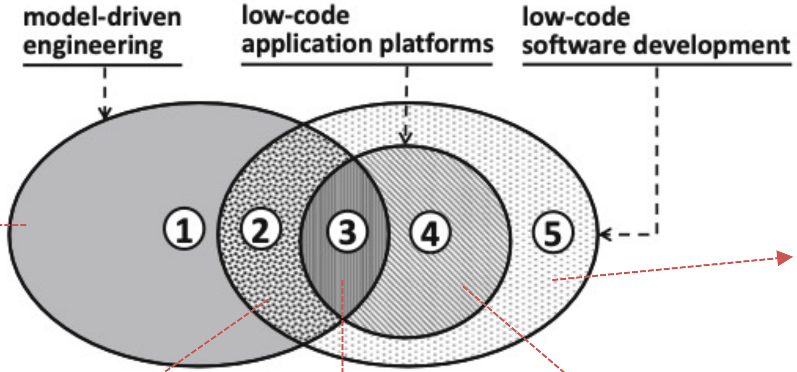
Criterion	LC_1	LC_2	LC_3	LC_4	LC_5	LC_6	LC_7	Overall
Static Perspective								
Mechanisms for data structure definitions	●●●	●●●	●●●	●●●	●●●	●●●	●●●	●●●
Data modeling component	●●●	●●●	●●●	●●●	●●●	●●○	○○○	●●○
Internal databases and persistence mechanisms	●●●	●●●	●●●	●●●	●●●	●●●	○○○	●●●
Access to external data sources (APIs)	●●●	●●●	●●●	●●●	●●●	●●●	●●●	●●●
Data reference models	●●○	●○○	●○○	○○○	●○○	●○○	○○○	●○○
Adaptation mechanisms for data (reference) models	●○○	●○○	●○○	○○○	●○○	●○○	○○○	●○○
Dynamic Perspective								
Mechanisms for program flow specifications	●●○	●●○	●●○	●●○	●●○	●○○	●●●	●●○
Process modeling component	●●○	●●○	●●○	○○○	●●○	●○○	●●●	●●○
Integration with static and functional components and artifacts	●●●	●○○	●●○	●●○	●●○	○○○	●●●	○○○
Process reference models	●○○	●○○	○○○	○○○	●○○	○○○	○○○	●○○
Adaptation mechanisms for process (reference) models	●○○	●○○	●○○	●○○	●○○	●○○	●○○	●○○
Functional Perspective								
Mechanisms for functional specifications	●●○	●●○	●●○	●●○	●●○	○○○	●●○	●●○
Functional modeling component	○○○	○○○	○○○	○○○	○○○	○○○	○○○	○○○
Generic functional reference specifications	●●●	●●●	●●●	●●○	●●●	●○○	●●○	●●○
Domain-specific functional reference specifications	●●○	●○○	●○○	○○○	●○○	○○○	○○○	●○○
Adaptation mechanisms for functional (reference) specifications	●○○	●○○	●○○	●○○	●○○	●○○	●○○	●○○
GUI Design								
GUI design component	●●●	●●●	●●●	●●●	●●●	●○○	●●●	●●●
Graphical GUI editor	●●●	●●●	●●●	●●●	●●●	●○○	●●●	●●●
Automatic generation of GUIs from data structures	○○○	●○○	●●○	○○○	●●●	●○○	●●○	●●○
GUI reference models	●●○	●○○	●○○	●○○	●○○	●○○	●○○	●○○
Roles and Users								
Specification mechanisms for roles and users	●●●	●●●	●●●	●○○	●●●	●●○	●●○	●●○
Modeling component for roles and users	○○○	●○○	○○○	○○○	○○○	○○○	○○○	○○○
Artificial Intelligence								
Internal artificial intelligence components	●●○	●○○	●●○	○○○	●●○	○○○	○○○	●○○
Integrability of external artificial intelligence services	●●○	●○○	●●○	○○○	●○○	○○○	○○○	●○○

Explanation: ○○○ = not or weakly addressed; ●○○ = partly addressed; ●●○ = well addressed; ●●● = extensively addressed.

LC1: Microsoft PowerApps, **LC2**: Mendix, **LC3**: Appian, **LC4**: Wavemaker, **LC5**: Pega, **LC6**: Quickbase, **LC7**: Bonita Platform

Low-Code *vs* Model-Driven Engineering

Source: Adapted from Di Ruscio et al. (2022).



Approaches that use **models** as machine-processable artefacts but **do not aim at reducing the amount of code**.

Approaches that use **models** as machine-processable artefacts and **aim to reduce the amount of code** required to implement a system (e.g. via code generation or interpretation) but **without offering deployment or life-cycle management capabilities**.

Platforms that use **models** to facilitate the development of software applications **with reduced code** and offer **built-in deployment and lifecycle management features**.

Platforms that **cannot be considered model-driven**, but facilitate the development of software applications **with reduced code** and **built-in deployment and lifecycle management features**.

Approaches that **cannot be considered model-driven**, but **aim to reduce the amount of code** required to implement a system **without offering deployment or lifecycle management capabilities**.

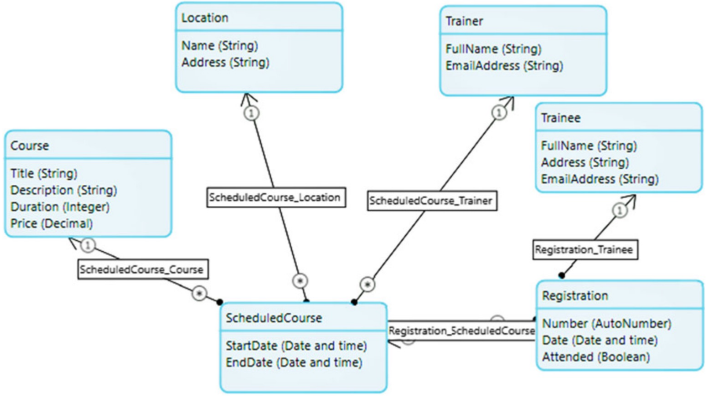
Low-code development:
Challenges and Opportunities

Limitations and challenges of low-code

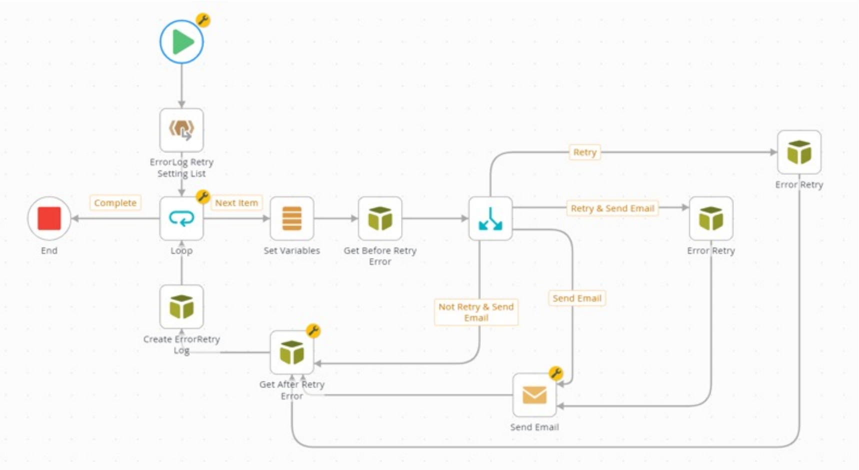
Practitioners' Perspective on Stack Overflow and Reddit (Luo et al., 2021).

Limitations and challenges of LCD	Example	Count
High learning curve	*you need to learn a lot about how this tool works to do the thing you're trying to do*	21
High pricing	*These larger vendors can get expensive, because they charge you for every user and you have to buy packages of 50 or 100 users*	13
Lack of customization	*Restrictive customisation on design and layouts*	11
Slow loading and publishing	*Loading speeds can be slow*	9
Less powerful than programming	*A full-fledged programming language will always have more power than a "no-code/low-code" solution such as PowerApps*	6
High complexity	*they're often too convoluted to use*	6
Complex issues still need coding	*If you go further and having a complex issue that can only be solved with invoking code or creating custom activities, you really need to code*	5
No access of source code	*Therefore you cannot take the code and use it elsewhere*	4
Not really ease of use	*No code is great, but not as easy as picking an app that's already written*	4
Limitation to experienced developers	*Most no-code tools are designed more like a prototyping tool and also targeted for non-developers which makes it very difficult for someone with development background to use*	4
Vendor lock-in	*Then there's the issue of vendor lock in. If you build using a nocode tool and they host etc. then if they raise their prices or shut down, that's going to a huge cost in downtime or rebuild and possibly lost data*	3
Difficulty of maintenance and debugging	*An additional risk is the continued support and maintenance of the low-code platform*	3
Difficulty of integration	*it looks to be a hard problem to make the UI, data store and calculations work together*	3
Unfriendly user experience	*it has a steeper and at times user unfriendly UX*	2
Need of basic programming knowledge	*most of them do require code at some point*	2

Issues: Model-based development



Domain model specification in Mendix



Workflow specification for error handling in Nintex K2

- **Proprietary notations** that are **not well-suited** for end users.
- Vendor **lock-in**.

Issues: Focus on application software

- Low-code platforms have been primarily focusing on **application software** for business processes automation of **low-to-moderate complexity**.
- This kind of application is considered *easy* to develop since they consist of **software-only solutions** that basically rely on manipulation of data forms.
- Noticeable **lack of support** for cyber-physical systems.

Perspectives on low-code

- **AI for low-code**
  - **Large language models** with code generation.
- **Communicating with humans and machines**
  - Low-code programs can serve both to communicate instructions to a computer and to **communicate among low-code users**.
  - Low-code can serve as a **lingua franca** to help citizen developers and pro-developers communicate more effectively with each other.

Perspectives on low-code

- **Domain-specific languages (DSLs)** for low-code
  - ❖ **Most VPLs are DSLs** (for example, Scratch), and both programming by demonstration and programming by natural language **usually target DSLs**. However, these DSLs are **not always exposed to the user**.
  - ❖ By exposing DSLs, users can more easily **read**, **test** and **audit** programs, **version** and **store** them in a shared repository, and **manipulate** them with tools for program transformation or generation.
  - ❖ **DSLs facilitate** program analysis, verification, optimization, parallelization, and transformation.

Perspectives on low-code

- **Combining** multiple **low-code techniques**
  - It can **compensate for the weaknesses** of individual techniques.
- **Meta-tools and meta-circularity**
  - A **meta-tool for low-code** is a tool used to implement low-code tools.
    - **Blockly**[1] is a tool for creating **VPLs**.
    - **DreamCoder**[2] is a tool for learning a library of reusable components along with a neural search policy for **PBE**.
    - **Overnight**[3] is a tool for building semantic parsers for **PBNL** with synthetic training data.

[1] Pasternak, E. et al. Tips for creating a block language with Blockly. In: Proceedings of Blocks and Beyond Workshop (2017); https://doi.org/10.1109/BLOCKS.2017.8120404.
[2] Ellis, K. DreamCoder: Bootstrapping inductive program synthesis with wake-sleep library learning. In: Proceedings of 2021 Conf. Programming Language Design and Implementation. 835–850; https://doi.org/10.1145/3453483.3454080
[3] Wang, Y. et al. Building a semantic parser overnight. In: Proceedings of the Annual Meeting of the Assoc. for Computational Linguistics. (2015), 1332–1342; https://www.aclweb.org/anthology/P15–1129.pdf

Perspectives on low-code

- **Meta-tools and meta-circularity**
  - ❖ A **meta-circular tool for low-code** is a meta-tool for low-code that is itself a low-code tool.
    - **VisPRO**[1] and **Racket**[2] are examples for creating VPLs.

[1] Zhang, K. et al. Design, construction, and application of a generic visual language generation environment. IEEE Trans Softw Eng. 27, 4 (2001), 289–307; https://doi.org/10.1109/32.917521.
[2] Andersen, L. et al. Adding interactive visual syntax to textual code. In: Proceedings of 2020 Conf. Object-Oriented Programming, Systems, Languages, and Applications; https://doi.org/10.1145/3428290.

EUP Learning Barriers

EUP Learning Barriers

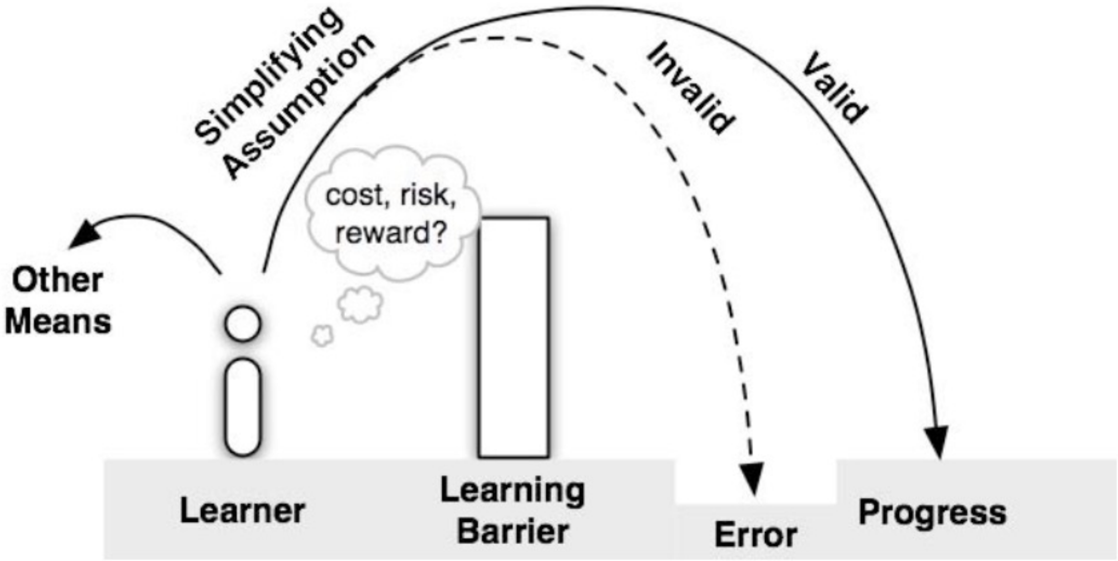
- End users are in constant need of **learning** how to operate and use the EUP system features.
  - This poses a potential **learning barrier**.
- From an *attention-investment perspective*[1], end users will **weigh the cost, risk, and reward** of overcoming the barrier.
- If the risk of failure outweighs the reward, end users **may abandon the system** for other tools.

[1] A. Blackwell, "First Steps in Programming: A Rationale for Attention Investment Models," IEEE Symposia on Human-Centric Computing Languages and Environments, Arlington, VA, Sept. 3-6, 2002, 2-10.

EUP Learning Barriers

- End users may also decide that **progress is worth the risk of failure**.
- In this case, they will make **several simplifying assumptions** about the EUP language, environment, and libraries in trying to acquire the necessary knowledge.
  - ❖ If their assumptions are **valid** with respect to the programming system, they will **make progress**.
  - ❖ If their assumptions are **invalid** – what's called ***knowledge breakdowns*** - they are likely to make an error.
- Learning barriers are defined as *aspects of a programming system or problem that are* ***prone to such invalid assumptions***.

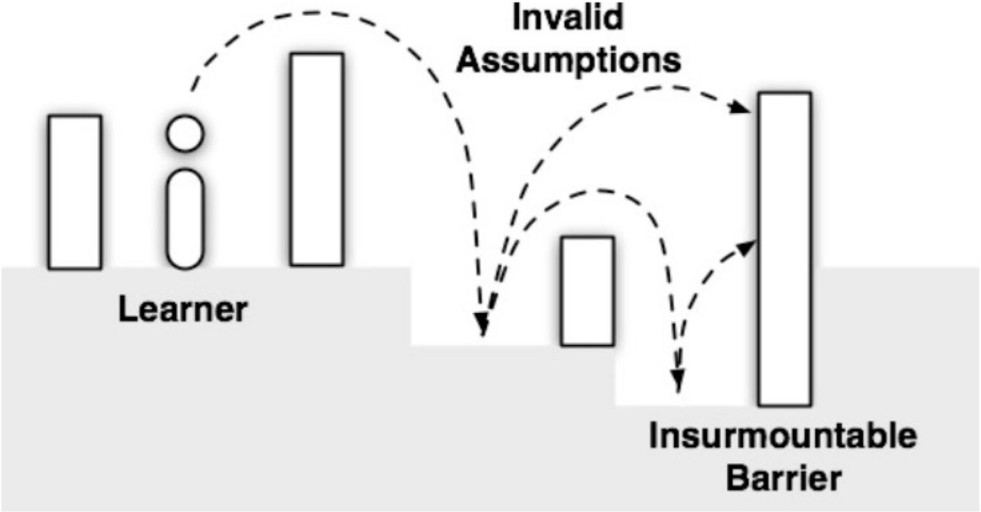
EUP Learning Barriers



(Ko et al. 2004)

EUP Learning Barriers

- End users can also reach ***insurmountable barriers*** which are properties of the EUP system or a programming problem that the the users could not understand **despite considerable effort**.



(Ko et al. 2004)

EUP Learning Barriers

- The **six learning barriers** proposed by Ko et al. (2004) are:
  - ❖ ***Design*** barriers
  - ❖ ***Selection*** barriers
  - ❖ ***Coordination*** barriers
  - ❖ ***Use*** barriers
  - ❖ ***Understanding*** barriers
  - ❖ ***Information*** barriers

Design Barriers

I don’t know what I want the computer to do...

Design Barriers

- ***Design barriers*** are **inherent cognitive difficulties** of a programming problem, **separate from the notation** used to represent a solution (i.e., words, diagrams, code).
- End users are **unable to conceive of a systematic way** to solve a problem.
    - ❖ E.g., when end users are unable to conceive of a systematic way to sort names and their best solution is *"just keep moving the names until it looks right!"*.
- Even when they are able to conceive a solution, they make **invalid assumptions** about their solution.
    - ❖ E.g., end users successfully test one cycle of their algorithm on a single data set on paper, and believed it to be correct for all.

Selection Barriers

I think I know what I want the computer to do, but I don’t know what to use...

Selection Barriers

- ***Selection barriers*** are properties of an environment's facilities for **finding what programming interfaces are available** and **which can be used** to achieve a particular behavior.
- These emerge when learners **could not determine which programming interfaces were capable** of a particular behavior.

Coordination Barriers

I think I know what things to use, but I don't know how to make them work together...

Coordination Barriers

- Coordination barriers are a programming system's limits on **how programming interfaces** in its language and libraries **can be combined to achieve complex behaviors** – what may be called "the invisible rules".
- End users encounter these **when they know what set of interfaces could achieve a behavior**, but **did not know how to coordinating them**.

Use Barriers

I think I know what to use, but I don't know how to use it...

Use Barriers

- ***Use barriers*** are properties of a programming interface that obscure:
  - ❖ **in what ways** it can be used,
  - ❖ **how to use** it, and
  - ❖ **what effect** such uses will have.
- These arose when end users **know what interface to use**, but were misled by these obscurities.

Understanding Barriers

I thought I knew how to use this, but it didn't do what I expected...

Understanding Barriers

- ***Understanding barriers*** are properties of a **program's external behavior** (including compile- and run-time errors) that **obscure what a program did or did not do at compile or runtime**.
- These emerged when end users **cannot evaluate their program's behavior relative to their expectations**.

Information Barriers

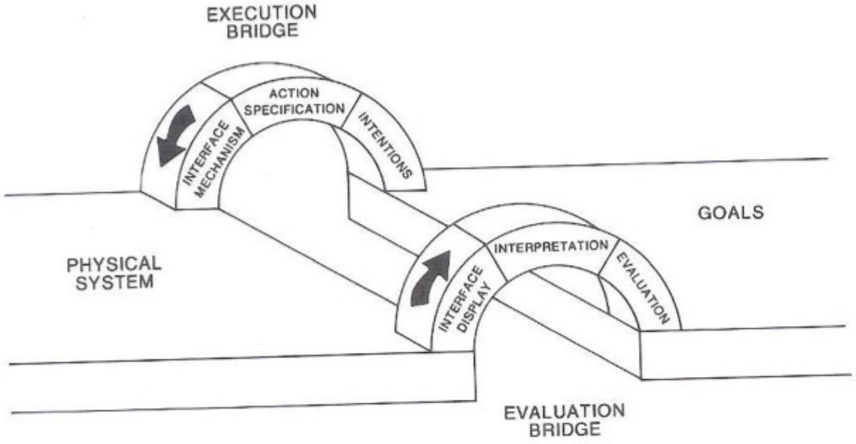
I think I know why it didn’t do what I expected, but I don’t know how to check...

Information Barriers

- ***Information barriers*** are properties of an environment that **make it difficult to acquire information about a program's internal behavior** (i.e., a variable's value, what calls what).
- These arise when **end users have a hypothesis** about their program's internal behavior, **but are unable to find or use the environment's facilities to test their hypothesis**.
  - ❖ E.g., when end users could not find the code that caused a particular error, they delete all of their recently modified code, confident that part of it must be guilty.

Relationship with Norman’s Gulfs

- The barriers share characteristics of Norman’s **gulf of execution** (the difference between users’ intentions and the available actions) and **gulf of evaluation** (the effort of deciding if expectations have been met).



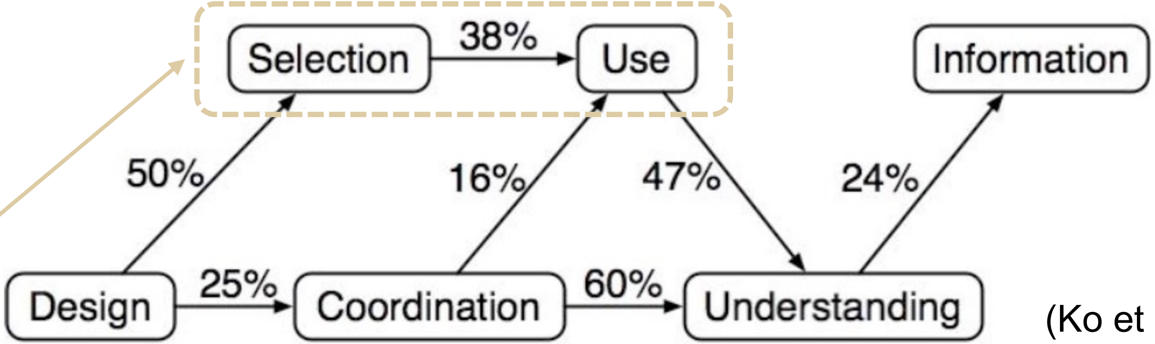
Relationship with Norman's Gulfs

- Three barriers pose **gulfs of execution** exclusively:
  - ❖ **Design**: mapping a desired program behavior to an abstract description of a solution.
  - ❖ **Coordination**: mapping a desired behavior to a computational pattern that obeys "invisible rules".
  - ❖ **Use**: mapping a desired behavior to a programming interface's available parameters.

Relationship with Norman's Gulfs

- Two pose **gulfs of execution and evaluation**:
  - ❖ **Selection**: mapping a behavior to appropriate search terms for use in help or web search engines, and interpreting the relevance of the results.
  - ❖ **Information**: mapping a hypothesis about a program to the environment's available tools, and interpreting the tool's feedback.
- **Understanding** barriers pose **gulfs of evaluation exclusively**, in interpreting the external behavior of a program to determine what it accomplished at runtime.
- Norman's recommendations on bridging gulfs of execution and evaluation **can be adapted** to programming system design.

How are the Learning Barriers Related?



(Ko et al. 2004)

Preventing invalid assumptions about programming interfaces' ***capabilities*** might avoid assumptions about their ***use***.

Challenges for more learnable EUP systems

- ***Design is inherently difficult***. To overcome **design barriers**, end users need **creativity**. **Programming systems should help scaffold creativity** with salient examples and other forms of inspiration.
- ***Finding behaviors is difficult***. To overcome **selection barriers**, **end users need help searching for behaviors**. Also, as more behaviors are offered, current tools will be increasingly ineffective.
- ***Invisible rules are difficult to show***. To overcome **coordination barriers**, **end users must know a programming system's invisible rules**. Today's systems lack explicit support for revealing such rules, merely implying them in error messages.

Challenges for more learnable EUP systems

- ***Textual programming interfaces are limited***. To avoid **use barriers**, **the feedback and interactive constraints of every programming interface must be carefully designed to match its semantics**. The textual, syntactic representations of today's systems make this goal difficult to achieve.
- ***Behavior is difficult to explain***. Overcoming **understanding and information barriers** requires some **explanation of what a program did or did not do**.

Further Reading

- Davide Di Ruscio, Dimitris Kolovos, Juan de Lara, Alfonso Pierantonio, Massimo Tisi, Manuel Wimmer (2022), ***“Low-code development and model-driven engineering: Two sides of the same coin?”***, Software and Systems Modeling, 21, pp. 437–446, Springer.
- Apurvanand Sahay, Arsene Indamutsa, Davide Di Ruscio, Alfonso Pierantonio (2020), ***“Supporting the understanding and comparison of low-code development platforms”***, In: 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA 2020), pp. 171-178, IEEE.

Further Reading

- Alexander C. Bock, Ulrich Frank (2021), ***"In Search of the Essence of Low-Code: An Exploratory Study of Seven Development Platforms"***, In: 2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), pp. 57-66, IEEE.
- Fulya Gürcan, Gabriele Taentzer (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.

Further Reading

- Yajing Luo, Peng Liang, Chong Wang, Mojtaba Shahin, Jing Zhan (2021), ***"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 2021), Article No.: 12, pp. 1–11, ACM.
- Amy J. Ko, Brad A. Myers, Htet Htet Aung (2004), ***"Six Learning Barriers in End-User Programming Systems"***, In: 2004 IEEE Symposium on Visual Languages - Human Centric Computing, pp. 199-206, IEEE.

Low-Code Development and EUP Learning Barriers

Thiago Rocha Silva (trsi@mmmi.sdu.dk)
Associate Professor