In-Situ Visual Programming

Ulrich Brandstätter
Software Competence Center Hagenberg GmbH (SCCH), Austria

Bernhard Schenkenfelder
Software Competence Center Hagenberg GmbH (SCCH), Austria

Abstract

Most Visual Programming Environments (VPEs) available today aim to make software development more accessible for specific domains, such as automation, business intelligence, data science, education, or real-time media processing. In their niches, VPEs offer several advantages over traditional text-based programming, including shorter training times, immediate visual feedback, and lower barriers to entry. With this work, we introduce *In-Situ Visual Programming (ISVP)*, a novel programming paradigm to enable users to create, modify, and contribute to software via visual programming in physical contexts. User-created and pre-built programs can be attached to and interlinked with physical objects – in an Augmented Reality (AR) environment. We believe that the spatial and contextual proximity of processing code and physical objects will make software development more intuitive, and we argue this position based on two model use cases.

2012 ACM Subject Classification Human-centered computing → Ubiquitous and mobile computing theory, concepts and paradigms

Keywords and phrases Visual programming, End-user programming, Programming paradigm

Digital Object Identifier 10.4230/OASIcs.Programming.2025.7

Funding The research reported in this paper has been funded by the Federal Ministry for Climate Action, Environment, Energy, Mobility, Innovation and Technology (BMK), the Federal Ministry for Labour and Economy (BMAW), and the State of Upper Austria in the frame of the SCCH competence center INTEGRATE (FFG grant no. 892418) in the COMET – Competence Centers for Excellent Technologies Programme managed by Austrian Research Promotion Agency FFG.

1 Introduction and Motivation

Visual Programming Environments (VPEs) allow software developers to write code using visual building blocks instead of, or in addition to, textual programming instructions. Instead of separate implementation files, module declarations, include directives, and function calls, these components are linked together using specific visual paradigms, such as edge connections for flow-based VPEs or jigsaw-like aggregations for block-based VPEs [3]. Either way, they promise a shallow learning curve and increased accessibility, especially when compared to traditional textual programming languages. They empower users of different backgrounds to contribute to and participate in the software development process. While people without a background or training in software development but with expertise in an application domain can benefit from the low barrier to entry of the visual paradigm, professional developers can be more productive [45].

In particular, multi-user capabilities for real-time collaboration allow many people to not only contribute ideas to creative tasks, but to work with the VPE at the same time, which has been shown to improve the quality compared to single-user systems. Working in parallel also makes visual programs more scalable [15]. In the context of learning to program, the benefits include a reduced collaboration effort and improved results, not only when groups work around shared tabletops, but also when using cross-device, co-located VPEs [40].

When Context-Oriented Programming (COP) was introduced more than two decades ago, the idea was to provide explicit support for spatial, temporal, or hardware-related domain-specific and technology-dependent attributes in programming languages. System designs in areas such as personalization, location awareness, and pervasive computing, where there is a great deal of variability due to the constantly changing properties, would benefit from not having to anticipate these dimensions of variability [21, 22].

By combining the above concepts, *In-Situ Visual Programming (ISVP)* takes software development to the next level. It is a novel programming paradigm based on real physical objects overlaid with and attached to virtual information. Their inherent spatial and contextual proximity is mapped in an AR-based environment, and the use of visual programming to manipulate these objects seamlessly blends into this paradigm. We see In-Situ Visual Programming as a concept with the potential to create a paradigm shift for collaborative software development. The simultaneous alignment and manipulation of physical and virtual objects with visual programming in AR environments can make software development processes more accessible and relatable, but also more playful and fun. Figure 1 illustrates the key motivation for ISVP:

- Users can attach programs directly to the physical objects on which they operate, rather than just abstracting their spatial relationships.
- Users can more accurately capture everything from requirements to results through an immersive experience with immediate feedback in the live domain.
- Flow-based visual programming has a low barrier to entry for software development, reducing the software development effort and facilitating the understanding of cause-and-effect chains through their visualization.
- Finally, by allowing multiple users to work either in parallel or collaboratively as they move through physical space, ISVP creates the conditions for engaging and fun virtual and real-world interactions.

The paper is organized as follows. In Section 2, we discuss related work, including both related scientific literature (2.1) and a review of VPEs (2.2). We then detail ISVP from a technical perspective in Section 3, followed by two use cases that illustrate how ISVP could be applied in these example domains (Section 4). We end the paper with our conclusions and ideas for future work (Section 5).

2 Related Work

Our vision of In-Situ Visual Programming (ISVP) is related to Visual Programming (VP), Low-Code Development (LCD), collaborative or multi-user software development, and immersive programming (Section 2.1) as well as context-oriented programming (Section 1). We also provide an overview of Visual Programming Environments (VPEs) that facilitate programming in XR or for XR (Extended Reality), which inspired our vision of ISVP (Section 2.2).

2.1 Literature Review

Common goals of **Visual Programming Environments (VPEs)** include increased accessibility, improved correctness, and improved code output performance [10]. VPEs involve a higher-level program description, and may make programming and debugging easier even for expert programmers [31]. They are often used in IoT and educational domains [26] as well

Figure 1 Following the ISVP paradigm, users can attach visual programs directly to physical objects through an immersive experience and collaboratively manipulate them.

as in more creative contexts [25]. VPEs can be used to better explain the inner workings of a system [28], to exchange information about software design [45], and to reduce the barrier to entry represented by syntax [30].

Specific VPE advantages are more explicit relationships [11, 8], immediate visual feedback [11], and the preservation of physical spatial relationships [28]. In literature, they are often considered an appropriate interface for parallel software development [8, 34] as well as for large-scale computing [37, 36]. Concerning teaching and learning scenarios, VPEs can be used to motivate students [31, 28], to more easily convey the joy of creating something [23], to attend more exploratory programming domains, e.g., by learning through play [4], and to convey computational thinking [27, 41]. Currently evolving interesting new application scenarios for VPEs include deep learning [42], Large Language Model (LLM) programming [46, 14], and (real-time) data science and machine learning [18]. A more traditional field of application where VPEs slowly start to shine concerns network programming [7, 35].

However, VPEs also involve several drawbacks. Concerning the interface, the limitations of physical screen space are often discussed in the context of VPEs [31, 11, 19], as are the difficulties with large programs and large data [3], also understanding visual programs is often considered challenging [31, 5, 3]. In terms of application areas, VPEs are often domain-specific, lack general functionality, and in many cases provide only a limited set of data types and relevant operators [31, 5, 3]. Additionally, many engineering problems are not inherently spatial [28], challenging the wide-ranging applicability of VPEs. Also, their overall efficiency is often brought into question [31, 11, 3], as is their portability and integrability with programs created in different languages [3]. Another downside concerns documentation, as visual programs are often considered poorly documented and unstructured [31, 11, 3]. Finally, the tools and user interfaces of VPEs are often seen as needing improvement, while at the same time VPEs are considered difficult to create and develop [28, 3].

Low-Code Development (LCD) is a human-centered programming paradigm that empowers end users without software engineering training to address their software requirements themselves. A Low-Code Development Platform (LCDP) helps users to increase their productivity and reduce the complexity of their tasks [13] by providing abstraction techniques that go beyond traditional textual programming [23, 33]. LCD has been around – under various names – for decades: Computer-Aided Software Engineering (CASE) in the 1980s, Rapid Application Development (RAD) in the 1990s, End-User Programming (EUP) in the 2000s, and Model-Driven Engineering (MDE) since 2000 are all aiming at reducing handwritten code in textual programming languages [16]. Many authors use the terms LCD and Visual Programming (VP) interchangeably [20, 38, 2, 9, 33]. However, in a study that examines LCD from a programming model perspective, Hirzel establishes a different taxonomy that groups the techniques of Visual Programming Languages (VPLs), Programming by Demonstration (PBD), and Programming by Natural Language (PBNL) side by side under the Low-Code umbrella [23].

Advantages of LCD that also apply to VP include the potential to alleviate the shortage of skilled personnel, to reduce or even avoid tedious tasks and the involved mistakes, and to multiply time-savings by deploying tasks to colleagues [12, 23] as well as their applicability for rapid prototyping [9]. In addition, a visual Low-Code Development Platform (LCDP) for generating Programmable Logic Controller (PLC) code is reported in an interview study as a common denominator used throughout the company. Intuitive and easy to understand, it allows the various departments involved, such as project planning, construction, and commissioning, to describe and discuss issues related to the machines represented in this visual language [39].

Multi-user and Immersive Programming. A study of remote pair programming and code comprehension reports that developers working in VR solved twice as many bugs in less time than programmers using a state-of-the-art screen-sharing system, highlighting both the effectiveness and efficiency of this paradigm [17]. Another study found that immersive authoring, i.e., creating VR content while being immersed in VR, can help users to prototype and modify programs by providing immediate feedback, thus reducing the learning curve, and some study participants found the dataflow authoring tool "fun and engaging to work with". Based on their findings, however, they do not see the benefit of giving users complete 3D freedom; instead, they suggest a combination of 2D and 3D interactions. First, nodes and operators should be connected on a 2D plane, and then they can in turn be connected to 3D objects [47]. This is also one of the core ideas of ISVP. Thomas et al. introduce the concept of *Situated Analytics* to facilitate sensemaking in the big data domain by associating information with physical objects, natural information exploration, and comprehensive information analysis. Many of their design considerations for the physical and logical worlds are also applicable to the programming of physical objects [43]. In an exploratory user study with seven expert users, Merino et al. observed highly interactive and engaging sessions. They found that the participants were willing to spend a lot of time, even on optional tasks, and were able to complete them while immersed. In addition, their implementation of an expressive yet simple language enabled the participants to take advantage of various visualization features. Again, some of these findings can be applied to the programming of physical objects [29].

2.2 Market Review

VPEs are frequently employed in conjunction with XR environments. Widely-used graphical programming environments, such as *vvvv*[1], a Visual Programming Environment for creative coding by the vvvv group, or *Blockly*[2], a popular web-based visual code editor by MIT and Google, often provide specific functions for XR use cases. *vvvv* for instance features a scene graph API, nodes for user tracking, a variety of input devices, and interfaces with several XR output devices. *BlocklyAR* [32] on the other hand is a research-driven AR extension for *Blockly*, it has the ambition to enable non-programmers to write AR applications using a web-based programming frontend. *SimpleAR* [1] is a high-level content design AR framework for users without programming knowledge; it underwent usability tests based on ISO 9241-11 and was found effective, efficient, and highly acceptable. All these examples use desktop-based development settings, i.e., although the results categorize as XR projects, they are created using 2D development environments.

Approaches that shift software development into XR environments are rarer, and these immersive development approaches primarily target VR. There are domain-specific applications, such as virtual robot programming. Using digital twins, sequences of movements are specified via motion tracking or inversed kinematics based on the spatial reference points given by the user [24]. Concerning XR, the available approaches can either be classified as brute-force or as experimental methods. Brute-force concepts normally use some form of desktop mirroring, with developers capturing the screens of their IDEs and rendering them on 2D VR planes. Although these virtual desktops suffer from various shortcomings of consumer-grade VR equipment, foremost the limited display resolutions, there is already a community sharing this ambition. Experimental general-purpose approaches originate from the worlds of gaming and research. Feature-wise, some of them are quite powerful, for example *RecRoom*[3], a VR multi-player online game by Against Gravity, features an in-world spatial VR programming language, whereas *Cubley* [44] focuses on Minecraft-style logic blocks. Of particular interest is *NeosVR*[4], a free-to-play VR multiplayer online app by Solarix, featuring multi-user capability on top of visual programming in VR.

3 ISVP Platform

The idea behind ISVP is to propose a novel programming paradigm that shifts development to AR environments. At the core of its implementation, the ISVP platform, is a flow-based visual language that allows multiple users to collaboratively develop programs that are directly attached to the real-world physical objects they interact with through an immersive AR experience. This section describes the underlying two-layer approach with a Visual Programming Kernel (VPK) as the base and an AR Visual Programming Interaction Frontend (ARF) on top.

The kernel implements a general-purpose, flow-based, high-performance, real-time visual programming language, including a scheduler, core and domain-specific nodes, graph handling, and multi-user support. To address several shortcomings of current VPEs discussed in literature while preserving their strengths, the authors present design considerations such as

[1] https://www.visualprogramming.net/
[2] https://developers.google.com/blockly/
[3] https://recroom.com/
[4] https://neos.com/

color-coded edge activation visualization, runtime deadlock visualization, a strongly typed input/output system with index sharing, edge data type limitation, user cursor visualizations, and session chat windows for the VPK in [6].

The frontend layer provides a set of functions to integrate the kernel, physical objects, and end users. First, a web-based user interface based on state-of-the-art technologies will be developed to visualize and modify the VPK in 2D, taking into account different devices (responsive design) and interaction modes such as click and touch. Then, detected and identified objects are visualized to anchor the VPK and assign specific functionality to these objects. And vice versa, the camera is used to select objects in the first place. In other words, the ARF provides AR-specific nodes for user and object tracking, multi-modal interaction, scene representation and rendering, and node visualization and manipulation tools.

The platform's modular kernel architecture allows the system to be extended with plug-ins, and extensions with third-party functionality can be realized by developers with access to the source code or plug-in API. The relevant functionality for both VPK and ARF is provided as nodes that contain related data, functions, input, and output. An illustration of the relevant components is shown in Figure 2. As we develop the ISVP platform, including both the VPK and ARF layers, the nodes will be determined by the requirements of the specific use cases provided by the industry-academic collaborations in which the authors are involved. The authors have implemented a VPK prototype following the dataflow programming paradigm and including an extensible stronly typed type system, a generalized UI widget system with relative and absolute anchoring, multi-layer serialization, a node registry supporting compile-time as well as run-time registration, drop-in profiling support, and real-time recursion detection [6]. However, the ARF is still in the design phase.

4 Use Cases

In this section, we present two model application domains, namely home automation and industrial training, which helped us to discuss, scope and design ISVP internally with fellow developers and externally with other stakeholders such as interested industry partners.

4.1 Home Automation

The first use case is home automation, where the goal is to determine whether the physical proximity of sensors, actuators, or displays and their actual programming is beneficial in practice. Sophisticated home automation systems often have centralized controls where users can change parameters and behaviors. Typically, they can be programmed via mobile applications or dedicated control units. As a result, there is a spatial discrepancy between the location of the physical object to be programmed and the location where the programming takes place, making it difficult to test and understand cause-and-effect chains. With ISVP, it is possible to align the programming with the objects. For example, all the effects of the light sensor in a hallway can be reviewed and modified in situ, bypassed altogether, or linked to additional conditions such as blinds or external factors such as sunlight or temperature. It also makes it easier for other people in the home to modify the home automation programming because the code is easier to find and, more importantly, to understand.

4.2 Industrial Training

The second use case is AR training scenarios. Especially in industrial contexts, AR training is often created by specialized companies or machine manufacturers. They are typically provided as ready-made scenarios with limited support for on-site modifications. With ISVP,

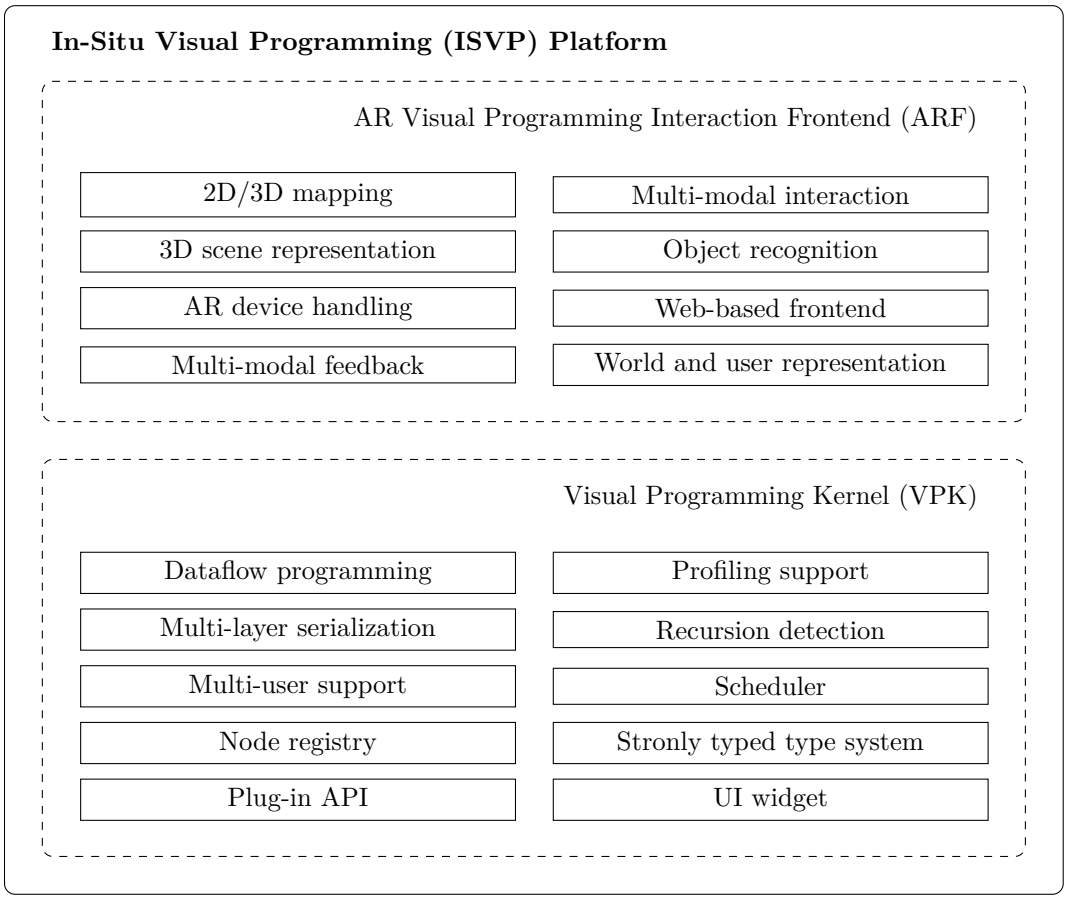

Figure 2 An overview of the ISVP platform with its main components and features.

it would not only be possible to modify such scenes according to individual needs, but also to create them from scratch. Existing production lines can be augmented with the virtual functionality relevant to training scenarios, such as playing safety training videos, simulating interaction consequences, emergency stop settings, simulated power failures, or visualizing production processes. For the creation and modification of AR trainings, we again consider the spatial proximity of an object and its programming as highly advantageous. Together with the accessible visual programming paradigm, it becomes easier for domain experts to share their knowledge. This use case could also include the ability to assess and react to the skills of training participants, as provided by the visual programming interface.

Given that digital twins not only know the state of the physical objects they mirror, but are also able to communicate back to set their state, the above use cases are also related to digital twins. When programming a physical device in context, e.g., a lamp as shown in Figure 1, ISVP needs to reflect not only the properties of the physical object for programming (i.e., $class = lamp, property = isOn$), but also the current state (i.e., $isOn = true$). Therefore, it is necessary to read the current state of the device in real time, which is usually provided by a digital twin.

5 Conclusions and Future Work

This vision paper outlines In-Situ Visual Programming (ISVP), a new programming paradigm that aims to move development into AR environments by providing a graphical frontend (AR Visual Programming Interaction Frontend, ARF) to a dataflow-oriented language (Visual Programming Kernel, VPK). ISVP increases the importance of spatial arrangements by using AR technology. Graphical programming blocks can be placed near and attached to physical objects, making the programming more intuitive and facilitating relatable cause-and-effect chains. These issues become even more interesting when considering ISVP's multi-user capability. As the physical space becomes one with the virtual programming canvas, users can arrange available building blocks, ready-made abstractions, and custom programs in 3D space. By enabling non-expert programmers to contribute to software development with a low barrier to entry, ISVP can alleviate the shortage of skilled workers in various domains. Users can more easily acquire programming skills and collaboratively adapt available solutions to individual needs.

The ISVP platform is being continuously developed as part of industry-academia collaborations in which the authors are involved. While a first prototype of the VPK is already under development, the ARF is still in the design phase. The future development of the ISVP platform is also accompanied by a scientific review that addresses research questions such as the following:

- How does the spatial proximity of physical objects and executable code make it easier to understand abstract concepts?
- How does the current state of the physical objects being programmed play a role in ISVP, and how can it be taken into account?
- How does ISVP foster collaboration between expert programmers, domain experts, and citizen developers?
- How does ISVP address the shortcomings of traditional visual programming environments?
- How does ISVP affect coordination and collaboration among programmers?
- How does ISVP make coding more fun?

References

1 Yuliana Apaza-Yllachura, Alfredo Paz-Valderrama, and Carlo Corrales-Delgado. Simplear: Augmented reality high-level content design framework using visual programming. In *2019 38th International Conference of the Chilean Computer Science Society (SCCC)*, pages 1–7, Concepcion, Chile, November 2019. IEEE. doi:10.1109/SCCC49216.2019.8966427.

2 Ryan Benac and Tauheed Khan Mohd. Recent trends in software development: Low-code solutions. In Kohei Arai, editor, *Proceedings of the Future Technologies Conference (FTC) 2021, Volume 3*, Lecture Notes in Networks and Systems, pages 525–533, Cham, 2022. Springer International Publishing. doi:10.1007/978-3-030-89912-7_41.

3 Sassi Bentrad and Djamel Meslati. Visual programming and program visualization – towards an ideal visual software engineering system –. *ACEEE International Journal on Information Technology*, 1:56–62, January 2011.

4 Mary Beth Kery and Brad A. Myers. Exploring exploratory programming. In *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 25–29, 2017. doi:10.1109/VLHCC.2017.8103446.

5 J.A. Borges and R.E. Johnson. Multiparadigm visual programming language. In *Proceedings of the 1990 IEEE Workshop on Visual Languages*, pages 233–240, 1990. doi:10.1109/WVL.1990.128412.

6 Ulrich Brandstätter, Bernhard Schenkenfelder, Doris Hohensinger, and Harald Kirchtag. Design considerations for a multi-user general-purpose flow-based visual programming environment. In *Proceedings of the 2024 International Conference on Advanced Visual Interfaces*, pages 1–3, Arenzano, Genoa Italy, June 2024. ACM. doi:10.1145/3656650.3656751.

7 Brian Broll, Ákos Lédeczi, Hamid Zare, Dung Nguyen Do, János Sallai, Péter Völgyesi, Miklós Maróti, Lesa Brown, and Chris Vanags. A visual programming environment for introducing distributed computing to secondary education. *Journal of Parallel and Distributed Computing*, 118:189–200, 2018. doi:10.1016/j.jpdc.2018.02.021.

8 James C. Browne, Syed I. Hyder, Jack Dongarra, Keith Moore, and Peter Newton. Visual programming and debugging for parallel computing. *IEEE Parallel Distrib. Technol.*, 3(1):75–83, March 1995. doi:10.1109/88.384586.

9 Alessio Bucaioni, Antonio Cicchetti, and Federico Ciccozzi. Modelling in low-code development: a multi-vocal systematic review. *Software and Systems Modeling*, 21:1959–1981, 2022. doi:10.1007/S10270-021-00964-0.

10 Margaret M. Burnett. *Visual Programming*, chapter 1, pages 275–283. John Wiley & Sons, Ltd, 1999. doi:10.1002/047134608X.W1707.

11 Margaret M. Burnett, Marla J. Baker, Carisa Bohus, Paul Carlson, Sherry Yang, and Pieter van Zee. Scaling up visual programming languages. *Computer*, 28:45–54, 1995. doi:10.1109/2.366157.

12 Rina Diane Caballar. Programming without code: The rise of no-code software development - ieee spectrum, March 2020. URL: https://spectrum.ieee.org/programming-without-code-no-code-software-development.

13 Jordi Cabot. 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*, pages 1–3, Virtual Event Canada, October 2020. ACM. doi:10.1145/3417990.3420210.

14 Yuzhe Cai, Shaoguang Mao, Wenshan Wu, Zehua Wang, Yaobo Liang, Tao Ge, Chenfei Wu, Wang You, Ting Song, Yan Xia, Jonathan Tien, and Nan Duan. Low-code llm: Visual programming over llms, 2023. doi:10.48550/arXiv.2304.08103.

15 J.D. Campbell. Multi-user collaborative visual program development. In *Proceedings IEEE 2002 Symposia on Human Centric Computing Languages and Environments*, pages 122–130, 2002. doi:10.1109/HCC.2002.1046364.

16 Davide Di Ruscio, Dimitris Kolovos, Juan de Lara, Alfonso Pierantonio, Massimo Tisi, and Manuel Wimmer. Low-code development and model-driven engineering: Two sides of the same coin? *Software and Systems Modeling*, 21(2):437–446, April 2022. doi:10.1007/s10270-021-00970-2.

17 James Dominic, Brock Tubre, Charles Ritter, Jada Houser, Colton Smith, and Paige Rodeghero. Remote pair programming in virtual reality. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 406–417, Adelaide, Australia, September 2020. IEEE. doi:10.1109/ICSME46990.2020.00046.

18 Ruofei Du, Na Li, Jing Jin, Michelle Carney, Scott Miles, Maria Kleiner, Xiuxiu Yuan, Yinda Zhang, Anuva Kulkarni, Xingyu Liu, Ahmed Sabie, Sergio Orts-Escolano, Abhishek Kar, Ping Yu, Ram Iyengar, Adarsh Kowdle, and Alex Olwal. Rapsai: Accelerating machine learning prototyping of multimedia applications through visual programming. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, CHI '23, New York, NY, USA, 2023. Association for Computing Machinery. doi:10.1145/3544548.3581338.

19 Mahmoud Fayed, Muhammad AL-Qurishi, Atif Alamri, M. Hossain, and Ahmad Al-Daraiseh. Pwct: a novel general-purpose visual programming language in support of pervasive application development. *CCF Transactions on Pervasive Computing and Interaction*, 2, August 2020. doi:10.1007/s42486-020-00038-y.

20 Henrique Henriques, Hugo Lourenço, Vasco Amaral, and Miguel Goulão. Improving the developer experience with a low-code process modelling language. In *Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, pages 200–210, Copenhagen Denmark, October 2018. ACM. doi:10.1145/3239372.3239387.

21 Robert Hirschfeld, Pascal Costanza, and Michael Haupt. *An Introduction to Context-Oriented Programming with ContextS*, volume 5235 of *Lecture Notes in Computer Science*, pages 396–407. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008. doi:10.1007/978-3-540-88643-3_9.

22 Robert Hirschfeld, Pascal Costanza, and Oscar Nierstrasz. Context-oriented programming. *Journal of Object Technology*, 7(3):125–151, 2008. doi:10.5381/JOT.2008.7.3.A4.

23 Martin Hirzel. Low-code programming models. *Commun. ACM*, 66(10):76–85, September 2023. doi:10.1145/3587691.

24 Michal Kapinus. *End-User Cobot Programming in Augmented Reality*. Ph.d. thesis, Brno University of Technology, Faculty of Information Technology, 2023. URL: https://www.fit.vut.cz/study/phd-thesis/891/.

25 Anastasia Kovalkov, Avi Segal, and Kobi Gal. In the eye of the beholder? detecting creativity in visual programming environments. *arXiv*, April 2020.

26 Mohammad Amin Kuhail, Shahbano Farooq, Rawad Hammad, and Mohammed Bahja. Characterizing visual programming approaches for end-user developers: A systematic review. *IEEE Access*, 9:14181–14202, 2021. doi:10.1109/ACCESS.2021.3051043.

27 Yustika Maharani, Cucuk Budiyanto, and Rosihan Yuana. The art of computational thinking through visual programming: A literature review. In *The 3rd International Conference on Science, Mathematics, Environment, and Education*, volume 2540, page 080036, January 2023. doi:10.1063/5.0105766.

28 Tim Menzies. Evaluation issues for visual programming languages. In *Handbook of Software Engineering and Knowledge Engineering*, May 2002. doi:10.1142/9789812389701_0005.

29 Leonel Merino, Boris Sotomayor-Gómez, Xingyao Yu, Ronie Salgado, Alexandre Bergel, Michael Sedlmair, and Daniel Weiskopf. Toward agile situated visualization: An exploratory user study. In *Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems*, pages 1–7, Honolulu HI USA, April 2020. ACM. doi:10.1145/3334480.3383017.

30 John T. Murray. Realityflow: Open-source multi-user immersive authoring. In *2022 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW)*, pages 65–68, 2022. doi:10.1109/VRW55335.2022.00024.

31 B. A. Myers. Visual programming, programming by example, and program visualization: A taxonomy. *SIGCHI Bull.*, 17(4):59–66, April 1986. doi:10.1145/22339.22349.

32 Vinh T. Nguyen, Kwanghee Jung, and Tommy Dang. Blocklyar: A visual programming interface for creating augmented reality experiences. *Electronics*, 9(8):1205, July 2020. doi:10.3390/electronics9081205.

33 Daniel Pinho, Ademar Aguiar, and Vasco Amaral. What about the usability in low-code platforms? a systematic literature review. *Journal of Computer Languages*, 74:101185, January 2023. doi:10.1016/j.cola.2022.101185.

34 José Quiroz-Fabián, Graciela Román-Alonso, Miguel Garcia, Jorge Buenabad-Chávez, Azzedine Boukerche, and Manuel Aguilar-Cornejo. Vppe: A novel visual parallel programming environment. *International Journal of Parallel Programming*, 47:1117–1151, December 2019. doi:10.1007/s10766-019-00639-w.

35 Elisa Rojas, Eder Ollora Zaballa, and Victoria Noci. Towards visual programming abstractions in software-defined networking. *Internet Technology Letters*, 5(3):e358, 2022. doi:10.1002/itl2.358.

36 Radosław Roszczyk, Marek Wdowiak, Michał Śmiałek, Kamil Rybiński, and Krzysztof Marek. Balticlsc: A low-code hpc platform for small and medium research teams. In *2021 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 1–4, 2021. doi:10.1109/VL/HCC51201.2021.9576305.

37 Kamil Rybiński, Michał undefinedmiałek, Agris Sostaks, Krzysztof Marek, Radosław Roszczyk, and Marek Wdowiak. Visual low-code language for orchestrating large-scale distributed computing. *J. Grid Comput.*, 21(3), July 2023. doi:10.1007/s10723-023-09666-x.

38 Apurvanand Sahay, Arsene Indamutsa, Davide Di Ruscio, and Alfonso Pierantonio. Supporting the understanding and comparison of low-code development platforms. In *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 171–178, August 2020. doi:10.1109/SEAA51224.2020.00036.

39 Bernhard Schenkenfelder, Christian Salomon, Georg Buchgeher, Robert Schossleitner, and Christian Kerl. The potential of low-code development in the manufacturing industry. In *2023 IEEE 28th International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–8, Sinaia, Romania, September 2023. IEEE. doi:10.1109/ETFA54631.2023.10275503.

40 Ben Selwyn-Smith, Craig Anslow, Michael Homer, and James R. Wallace. Co-located collaborative block-based programming. In *2019 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 107–116, 2019. doi:10.1109/VLHCC.2019.8818895.

41 Karin Stolpe and Jonas Hallström. *Visual Programming as a Tool for Developing Knowledge in STEM Subjects: A Literature Review*, pages 130–169. Brill Academic Publishers, January 2024. doi:10.1163/9789004687912_007.

42 Srikanth G Tamilselvam, Naveen Panwar, Shreya Khare, Rahul Aralikatte, Anush Sankaran, and Senthil Mani. A visual programming paradigm for abstract deep learning model development. In *Proceedings of the 10th Indian Conference on Human-Computer Interaction*, IndiaHCI '19, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3364183.3364202.

43 Bruce H. Thomas, Gregory F. Welch, Pierre Dragicevic, Niklas Elmqvist, Pourang Irani, Yvonne Jansen, Dieter Schmalstieg, Aurélien Tabard, Neven A. M. ElSayed, Ross T. Smith, and Wesley Willett. *Situated Analytics*, volume 11190 of *Lecture Notes in Computer Science*, pages 185–220. Springer International Publishing, Cham, 2018. doi:10.1007/978-3-030-01388-2_7.

44 Juraj Vincur, Martin Konopka, Jozef Tvarozek, Martin Hoang, and Pavol Navrat. Cubely: virtual reality block-based programming environment. In *Proceedings of the 23rd ACM Symposium on Virtual Reality Software and Technology*, pages 1–2, Gothenburg Sweden, November 2017. ACM. doi:10.1145/3139131.3141785.

45 Bianca Wiesmayr, Alois Zoitl, and Rick Rabiser. Assessing the usefulness of a visual programming ide for large-scale automation software. In *2021 ACM/IEEE 24th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, pages 297–307, 2021.

46 Tongshuang Wu, Ellen Jiang, Aaron Donsbach, Jeff Gray, Alejandra Molina, Michael Terry, and Carrie J Cai. Promptchainer: Chaining large language model prompts through visual programming. In *Extended Abstracts of the 2022 CHI Conference on Human Factors in Computing Systems*, CHI EA '22, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3491101.3519729.

47 Lei Zhang and Steve Oney. Studying the benefits and challenges of immersive dataflow programming. In *2019 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 223–227, Memphis, TN, USA, October 2019. IEEE. doi:10.1109/VLHCC.2019.8818856.