Assessing the Usefulness of a Visual Programming IDE for Large-Scale Automation Software

Bianca Wiesmayr*, Alois Zoitl*†, *Member, IEEE*, Rick Rabiser*†
†*CDL VaSiCS*, **LIT CPS Lab, Johannes Kepler University Linz, Austria*
{bianca.wiesmayr, alois.zoitl, rick.rabiser}@jku.at

***Abstract*—Industrial control applications are usually designed by domain experts instead of software engineers. These experts frequently use visual programming languages based on standards such as IEC 61131-3 and IEC 61499. The standards apply model-based engineering concepts to abstract from hardware and low-level communication. Developing industrial control software is challenging due the fact that such systems are usually one-of-a-kind systems that have to be maintained for many years. These challenges, together with the growing complexity of control software, require very usable model-based development environments for visual programming languages. However, so far only little empirical research exists on the practical usefulness of such environments, i.e., their usability and utility. In this paper, we discuss common control software maintenance tasks and tool capabilities based on existing research and show the realization of these capabilities in 4diac IDE. We first performed a walkthrough of the demonstrated capabilities using the cognitive dimensions of notations framework from the field of human-computer interaction. We then improved the tool and conducted a user study involving ten industrial automation engineers, who used 4diac IDE in a realistic control software maintenance scenario. Our findings demonstrate how the usefulness of IDEs can be successfully investigated using a multi-phase approach that includes a walkthrough and a user study. We discuss lessons learned and derive general implications with respect to large-scale applications for developers of IDEs that we deem applicable in the context of (visual) model-based engineering tools.**

***Index Terms*—Usefulness study, Open source software, IEC 61499, Modeling tools, Model-driven engineering**

I. Introduction

Visual programming languages can improve the communication and collaboration between software developers. In comparison to textual languages, visual representations can stimulate more active discussions and improve the memorability of design details [1]. Additionally, the subjective satisfaction of developers can be higher for visual notations, as Meliá et al. [2] showed in their study in which students performed maintenance tasks on software models in both a textual and a visual notation. The measured efficiency and effectiveness of the performed tasks were, however, reduced in the visual model.

Several visual languages are well-established in their respective domains. LabView [3] is a language targeted at data acquisition and manipulation. Simulink [4] is a block-based language that is used for control engineering. Domain-specific languages (DSLs) for control software in automated production systems are defined in industrial standards and are the focus of our work. IEC 61131-3 standardizes three visual languages: (1) Ladder Diagram resembles electrical plans, (2) Sequential Function Chart (SFC) is a state diagram, and in a (3) Function Block (FB) Diagram, blocks are connected via data signals [5]. An event-triggered Function Block Diagram is defined in IEC 61499 [6]. Algorithms in textual language can be integrated in these models. Control software is developed based on industrial standards (i.e., ISO/IEC) because of the long life cycles of production plants [7]. These languages are targeted at automation engineers, not software engineers, and abstract the control logic from hardware and low-level communication. As requirements for control software are derived from electrical and mechanical plans, the DSLs are optimized for matching the mental model of automation engineers.

Several challenges arise when developing industrial control software: automated production systems are produced as one-of-a-kind and are tailored to the needs of a customer. Hence, reusing control software is challenging and requires extensive support for managing variability [8]. Furthermore, the life cycles of the physical equipment by far exceed those of the software: control software has to be evolved over decades [9]. The requirements for modern automated production systems lead to growing complexity of control software: an increasing number of interacting (cyber-)physical components, complex communication between components, and unwanted physical effects that become more relevant due to the high accuracy that is required [10]. Regarding the usability of tools, it has to be considered that automation engineers tend to have a strong background in electrical and mechanical engineering, but not in software engineering [9]. As the control application is typically finalized only at the factory or plant during commissioning of the machine, further user groups are involved [9].

IDEs for block-based software have been evaluated mostly with a focus on creating new software, but not on software maintenance. Only little empirical research exists regarding their usefulness in practical environments and for industrial users. Usefulness regards a tool's utility, i.e., to what degree its functionality allows users to do what is needed, and its usability, i.e., how well users can exploit the offered functionality [11], [12]. Assessing usefulness requires qualitatively studying users and their behavior [13]. A study covering programming editors that support Scratch and related languages revealed usability flaws that are relevant for any visual language, but was conducted only with students of a single discipline [14]. Several studies evaluated the programming languages that are relevant for industrial automation. In a

multi-modal usability study of the IEC 61499-IDE Eclipse 4diac, various editors and views were evaluated from the perspective of a broad user group. However, this study did not cover the handling of large-scale applications [15]. Obermeier et al. [16] have compared a Function Block Diagram to a modeling approach utilizing simplified variants of UML diagrams. Their study focused on new applications and was performed with a large group of participants consisting of both students and industrial practitioners. The visual language SFC was compared to the UML Activity Diagrams and State Charts in [17] with a focus on process technology, showing that Activity Diagrams are best suited for designing flexible and modular sequences. The study evaluated the languages based on the cognitive effectiveness, but does not include experimental results. We therefore conclude that existing user studies do not evaluate the usability of handling large-scale automation software in a realistic maintenance scenario.

Empirical studies can increase the acceptance of tools in industry, as they help engineers to select and adapt tool capabilities that are relevant for their application context [18]. Following this goal, this paper provides the following *contributions*: (i) we discuss common control software maintenance tasks and tool capabilities based on existing research. (ii) We show the realization of these capabilities in 4diac IDE [19] and assess them in a walkthrough using the Cognitive Dimensions of Notations (CD) Framework [20]. (iii) Based on the findings of this assessment, we conducted a usefulness study involving industrial automation engineers, who used 4diac IDE in a realistic control software maintenance scenario. (iv) We discuss lessons learned and derive general implications for developers of IDEs for visual languages.

II. Background

We first briefly describe common scenarios for developing and maintaining control software. We further describe existing IDEs for control software that support these tasks.

A. Control software development and maintenance tasks

During the development of production systems, even in late stages such as commissioning, frequent adaptations of the software are required to address changing requirements [21]. In addition to creating new software, maintenance of existing software is highly relevant, particularly in automation engineering. Industrial production plants have life cycles of several decades. During this time, the software is typically updated more frequently than the hardware, roughly every six to twelve months [9], to allow adaptations of the production process when additional products are manufactured and to benefit from technological advances [21].

Common tasks for control software maintenance have been discussed in the literature. For instance, typical programming tasks for machine and plant automation were identified by Obermeier et al. [22] to form a basis for usability studies in the domain. The authors suggest an adequate task complexity to limit the number of programming errors, while still receiving sufficient feedback. The resulting task descriptions cover requirements elicitation, identifying interfaces to the environment, and implementing the actual system functionality. Tasks for evolving automation software were discussed by Legat et al. [23]. Relevant scenarios include adding new components that operate in parallel to increase the capacity, adding new variants of supply material, introducing redundancy to improve reliability, or replacing mechanical submodules. Another typical workflow involves reusing legacy control software. First, code fragments that are suitable for reuse are identified. Adaptions are typically needed to ensure that the code is sufficiently abstract and can be parameterized if variability has to be considered. Finally, the code fragment is stored in a library for reuse [24].

B. IDEs for control software

IEC 61499 is a domain-specific modeling language (DSML) and well-suited to support developers and maintainers in the tasks described in the previous section. Several actively maintained IDEs support IEC 61499-models [25], [26]:

- NxT Technology IDE from nxtControl GmbH [27]
- ISaGRAF Workbench from Rockwell Automation [28]
- FBDK from Holobloc Inc. [29]
- 4diac IDE, an open source project that is hosted by the Eclipse foundation [19]

The offered assistance during development and maintenance varies among these tools. Furthermore, some tools extend the language: for instance, NxT Technology IDE [27] supports comment areas and special automation components that include support for human-machine-interaction (HMI). 4diac IDE [19] supports aggregating FB instances without affecting the library, which is recommended for large-scale applications [30]. We use 4diac IDE as a tool environment for our study, as it is based on widely applied Eclipse technologies for creating DSML editors. Some usability flaws we identify, which are related to the underlying Eclipse platform, may thus also affect other modeling tools that are based on it. As it is available as an open source project, we can extend 4diac IDE with plug-ins and provide bug fixes.

III. Research Approach

Our study investigated two research questions on the usefulness of control software development tools' capabilities as implemented in the tool 4diac IDE:

- RQ1 What is the usability of the tool capabilities for maintaining an unknown and complex software?
- RQ2 What is the utility of the tool capabilities for maintaining an unknown and complex software?

For that we assumed the following maintenance of (legacy) control software setting: maintenance work is required in an existing plant. The motor of one conveyor belt is broken and needs to be replaced. As the installed model is no longer available, a newer version of the motor has to be installed. Therefore, adaptations to the control software are required. We designed the maintenance tasks based on the identified scenarios (cf. Section II-A) to reflect realistic practical settings.

Regarding *RQ1*, we assessed the tool capabilities implemented in 4diac IDE from the perspective of industrial end users, guided by the CD framework and Nielsen's usability dimensions [12]. Regarding *RQ2*, we investigated whether users can successfully perform maintenance tasks using 4diac IDE and how they perceive the usability of the tool. We also collected the perceived opportunities and risks [31] of using 4diac IDE in practice.

1) Preparation and Initial Assessment: We first analyzed existing tools and the literature to distill common tasks and tool capabilities for end users in typical control software maintenance scenarios. We also discussed how the capabilities are realized in 4diac IDE. We then assessed 4diac IDE using the CD framework [20] to reveal usability flaws that could bias the study with industrial end users. The CD framework is well known in the human-computer interaction (HCI) community and has been used successfully to assess software engineering tools in the past [14], [18], [32]. It supports evaluating and characterizing capabilities of interactive artifacts, such as software tools, and offers a vocabulary for discussing trade-offs based on a set of basic user activities and cognitive dimensions. Specifically, we performed a walkthrough of 4diac IDE based on typical control software maintenance tasks and tool capabilities, to reveal usability issues requiring tool improvements before the actual study with engineers. We addressed potential showstoppers by adapting 4diac IDE.

2) Usefulness Study: We first defined the study method based on our findings from the CD assessment, following the guidelines for conducting empirical studies described by Runeson and Höst [33] and Ko et al. [34]. We selected the study system and subjects (ten industrial automation engineers of our industry partner). We also defined the experimental setting, the maintenance tasks to be conducted by subjects, the data sources and collection methods, as well as the data analysis and reporting process (cf. Section VI). Before the actual study, we conducted pilot experiments with three participants (two students and an industrial expert), to reveal potential flaws in the designed tasks or bugs that could influence the results of the study. All subjects were asked to watch an introductory video to the tool before the study. The user study was conducted remotely via Zoom, separately with each subject. During the study, each subject used 4diac IDE (version 1.14.0 RC1) to maintain an IEC 61499 application (adapted from [35]). We asked each subject to "think aloud" [11], i.e., to describe what they were doing and to express any concerns. After the working session with the tool, we performed a semi-structured utility interview [31] and asked the subjects to fill in a usability questionnaire [12]. We discuss results in relation to the cognitive dimensions (cf. Section VII).

IV. 4diac IDE and IEC 61499

Eclipse 4diac [19] implements a tool environment for the visual modeling language that is defined in the industrial standard IEC 61499 [6] and is targeted at domain experts, i.e., automation engineers. In this section, we will first summarize the basics of the language, and then the tool environment.

A. The domain-specific modeling language IEC 61499

IEC 61499 allows engineers to model a distributed automation system. The modeling language includes an application model with the control software, and a system configuration model that captures the hardware and its network. Fig. 1 shows the relation between these two models. The application model is independent of the hardware, which is a key advantage of IEC 61499 over programming languages that are already established in the domain. The standard IEC 61499 defines a set of FB types, which constitute a library with the most important functionalities. The internal functionality of custom FB types can be implemented either in a visual or in a textual language. FB types can be instantiated in the application.

The language is strongly typed, i.e., an event type or data type is assigned to each pin of an FB. Control software is furthermore developed for repeated execution. All FB instances in the application were traditionally executed periodically, independent of external triggers. In contrast, IEC 61499-models are event-triggered and are executed on demand: whenever an event arrives at the input pin of an FB instance, this FB is executed following a run-to-completion semantics. If necessary, subsequent application parts are triggered by sending one or more output events. The internal state of an FB instance persists between executions.

The DSML supports concepts that are known from object-orientation, including abstract interface definitions, types and instances, and encapsulation. IEC 61499 has dedicated elements to structure models hierarchically: subapplications (subapps) for grouping FB instances, and adapters for grouping interface pins. The interface definition of subapp types is identical to the one of FB types, but their behavior is exclusively defined by the encapsulated application part. Any number of hierarchical levels is possible because subapps may themselves be composed of instances of both FBs and other subapps. Unlike FBs, subapps may be distributed across devices.

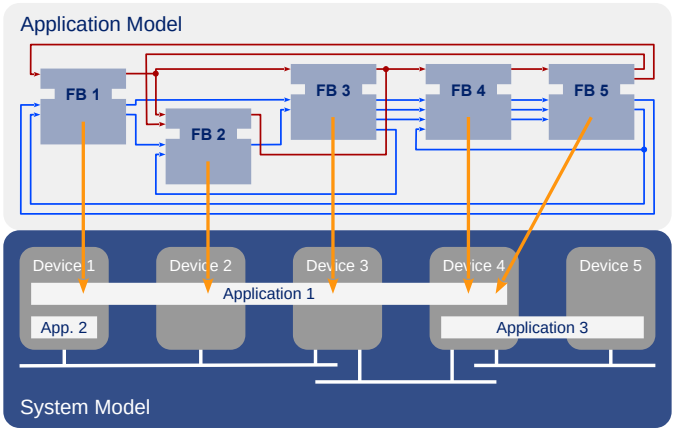

Fig. 1. Core models defined in IEC 61499. The software is described in the block-based application model (top), where each block type (Function Block, FB) offers and encapsulates a certain functionality. In the application, FB types are instantiated and wired together to form a network: the FB instances exchange information via event (red) and data connections (blue). Each FB has a well-defined interface with input and output pins. For the execution, the application is distributed across devices from the System model (bottom).

B. Development Environment under Test: 4diac IDE

Eclipse 4diac is an open-source environment for modeling systems based on IEC 61499 that includes both an IDE and a runtime environment for executing IEC 61499-applications. The 4diac IDE is structured into five views (cf. Fig. 2). It is developed in Java and Xtend as a set of plug-ins for the Eclipse platform, using technologies that are commonly applied for editors of DSLs: an EMF metamodel, XText parsers, and the graphical editing framework (GEF3) [36]. The runtime is programmed in C++ and can be executed on various devices, from low-cost platforms, such as Raspberry Pi, to industrially relevant Programmable Logic Controllers (PLCs) and industrial PCs. The runtime thus introduces an abstraction layer between the application model and the hardware.

Industrial automation engineers can particularly benefit from advanced IDE capabilities when working with large-scale applications. The tool needs to provide high performance for navigating through applications with thousands of instances. Furthermore, information has to be well accessible, so that information about the environment of a component can be discovered and inconsistencies detected easily. The variability of production plants furthermore results in high requirements and challenges for reusing application parts. Vendor-neutral solutions are thus preferable to fully benefit from the hardware-neutral design of applications in IEC 61499.

C. Assessed tool capabilities

The goal of this study is to evaluate the usability and utility of tool functionality for large-scale applications. Typical tasks of industrial developers are outlined in Section II-A. Developing new FB types is outside the scope of our study, which focuses on editing (maintaining) existing application models using a pre-defined library of FB types. Based on our analysis, we choose the following tasks for our user study. We created a video for demonstrating these tasks in 4diac IDE [37].

1) Orienting in an unknown application: Industrial automation engineers frequently have to perform maintenance tasks directly on-site at the machine. In this situation, the engineer has to quickly navigate through a partly unknown control application. In our study, we therefore ask subjects to (i) find the motor that will be replaced, (ii) find out whether there are other motors in the application, and (iii) follow an event connection to identify which application part is triggered next.

2) Creating/Removing hierarchies: Following a workflow for reusing legacy software [24], subjects group existing control code in a subapp and name it with a valid IEC 61499 identifier. They remove a needless grouping and adjust the scope of their subapp.

3) Working with the library: Subjects save their subapp (from task 2) to the library for later reuse. They add template code for the new motor to the library from outside the tool. Subjects then replace their subapp with an instance of this new type, while keeping the connections intact.

4) Editing: Subjects now detype their motor subapp and perform edits. In this task, we analyze general editing features, i.e., extending the interface, adding new FB instances of existing types from the library, as well as editing and adding connections.

V. Cognitive Assessment of 4diac IDE

As preparation for the user study, we assessed the capabilities of 4diac IDE (regarding the four tasks described above) using the CD framework [20]. The CD framework differentiates four basic types of user activities: incrementation, transcription, modification and exploratory design [18]. Maintaining software in an IDE for a visual programming language is related with exploratory design as it combines incrementation with modification without knowing the desired end state in advance: adding new elements (e.g., FB instances, connections) to the application can be considered an incrementation, as it adds further information without altering the existing application structure. Adding/removing hierarchies or layouting an application can be considered a modification.

Each user activity involves usability trade-offs regarding one or several cognitive dimensions. For example, a high viscosity, i.e., the resistance to change, is harmful for modification and exploration activities, but has less impact on the one-off tasks performed in transcription and incrementation. Fourteen cognitive dimensions were initially defined for the CD framework. Thirteen of these are relevant for exploratory activities and thus also in our context, as defined in the CD framework. For each of the four maintenance tasks described in Section IV-C, we analyzed how well 4diac IDE addresses the relevant cognitive dimensions. Some dimensions are relevant for all activities and thus crosscut our structure. Specifically, our aim was to reveal potential showstoppers for each task, which could inhibit the successful completion of the user study. We tested and analyzed 4diac IDE based on our defined tasks and the CDs to reveal such errors, but also considered prior experiences with users. In our discussion below, we highlight such cases with the keyword `FX`, indicating that we fixed and improved 4diac IDE before involving industrial users. The label `OK` highlights dimensions that we considered sufficiently supported according to the CD framework. We did not focus our user study on these dimensions. All other paragraphs, highlighted with `ST`, describe dimensions which we need to investigate more closely in our user study by refining our research method accordingly. Table I provides an overview of the assessed activities, 4diac IDE's tool support, relevant cognitive dimensions, and the assessment results. It also previews whether the user study eventually revealed a dimension to be well supported (+), well supported but with potential for improvement (+/−), or not well supported and the tool needs to be improved (−). Detailed results are reported in Section VII.

A. Orienting in an unknown application

The System Explorer (cf. Fig. 2) view shows all instance names of an application as a tree. Each element can be selected

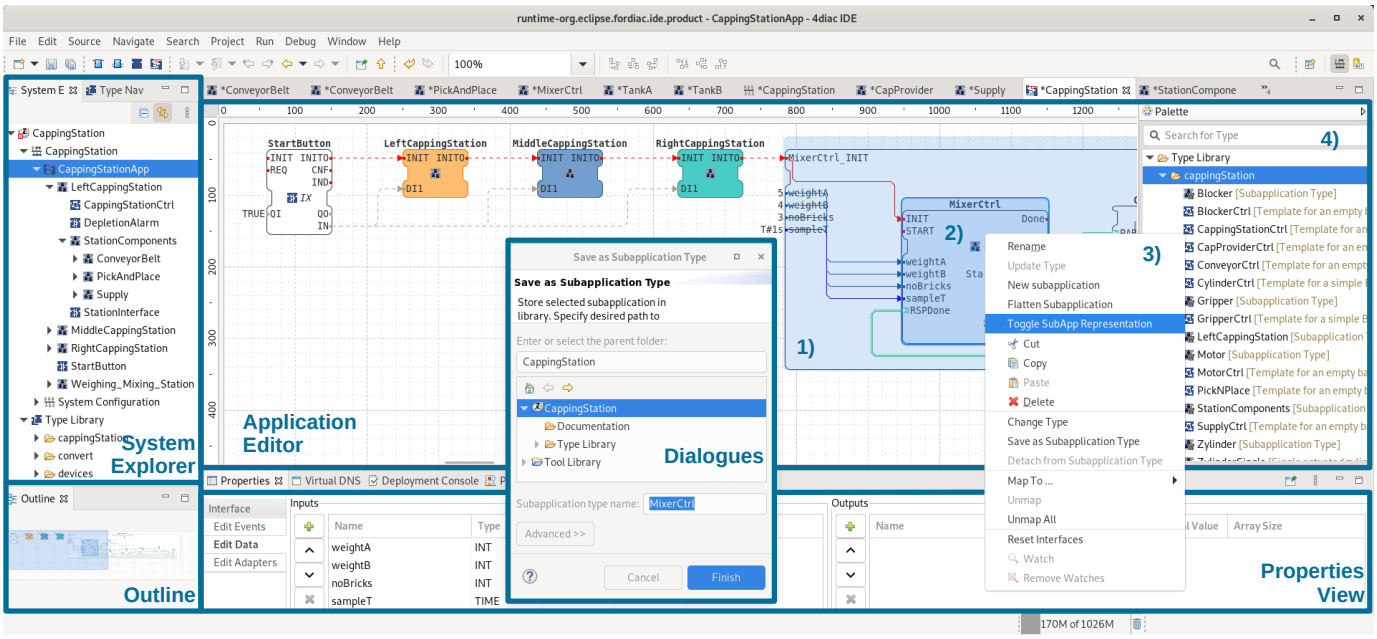

Fig. 2. Compilation of the main views and editors provided by 4diac IDE for developing IEC 61499-based automation solutions: (1) expanded subapplication with hover feedback, (2) selection feedback, (3) context menu, and (4) palette for adding FB instances.
The *System Explorer* on the left lists all projects and their contents. Each project comprises (i) the IEC 61499-system model and (ii) a project library containing FB types that are either defined in the standard or created by a developer for the project. An overview of the system model is provided as a tree that shows all applications with their full hierarchy and all instances of FBs, as well as the system configuration with all devices and resources. This tree view allows the developer to get an overview of the system model but only provides information on the hierarchy between FBs, but not connections between FB instances. From the *System Explorer*, users can navigate to the corresponding location in the graphical *Application editor*. In this editor, the network of FBs is shown, new instances can be added, and connections can be added or reconnected. Individual items can be modified in the *Properties View*. When an FB instance is selected, its settings are shown in several tabs and include the instance name, the descriptions of an instance, and its interface. Information that is defined by the type is provided as read-only. This includes the type description, version information, and the interface of the FB. The *Outline* allows navigating and orienting in large applications as it provides an overview of the full drawing area. Some editing operations, such as creating new types, involve *Dialogues*.

and opened (via double-click) in the graphical editor. Selected elements are highlighted and their attributes are shown in the Properties view. The Outline (cf. Fig. 2) provides an overview ("minimap") of the diagram that is opened in the graphical editor and also allows navigating in the diagram.

1) CD assessment: For orienting in unknown applications, developers can rely on *Secondary Notations* that 4diac IDE provides: instance names, instance comments, as well as type names and type descriptions indicate the functionality of an FB type or the role of its instance (OK). FB instances, however, do not visually represent their behavior and therefore have a low *Role Expressiveness*. For example, an FB adding two numbers could be represented graphically with a mathematical symbol (ST). Subapps group FB instances logically and increase *Abstraction* (OK), but reduce the *Visibility* of application parts (FX): when application parts are structured into a subapp, they cannot be viewed in their context anymore. Only the contents of a single (sub-)application is shown in the graphical editor. Several graphical editors can, however, be opened in parallel and arranged freely to compare application parts (*Juxtaposability*, OK). In large (sub-)applications, we identified that a selected FB instance and its connections are difficult to find, although they are highlighted by a border (*Diffuseness*, FX). Finally, the IDE does not provide any navigation along connections, although developers may need to follow a signal path across hierarchical levels (*Hard Mental Operations*, FX).

2) Tool improvements: We enhanced the graphical editor with additional mechanisms to navigate along dependencies and between hierarchies. A new feature for expanding a subapp (cf. element 1 in Fig. 2) allows viewing its contents as part of the surrounding FB network and thus increases visibility. We improved the highlighting of selected FB instances with a blue overlay, which resembles the highlighting for text. A transparent overlay is shown already upon hovering over FBs and pins, thus better visualizing the objects that are available for user interaction (cf. elements 1 and 2 in Fig. 2).

B. Creating/removing hierarchies

Subapps and adapters group elements to a hierarchical structure. FB instances that are contained in a subapp communicate with the external network via a dedicated interface. Developers can design new subapps either bottom-up with a self-defined interface, or create them top-down from an existing network of FBs. For the latter, the tool infers the required interface from the existing connections. FB instances can be added to alter the scope of a subapp. If required, 4diac IDE updates the subapp interface automatically. A subapp can also be flattened, i.e., deleted and replaced by its contents.

1) CD assessment: Untyped subapps (cf. Section V-C) have a low *Viscosity* as they can be easily created from a network of FBs (OK). As the subapp interface is updated automatically, also the *Error-Proneness* is reduced (OK). However, the reverse

operation of moving an FB instance to the parent network is not supported (*Premature Commitment*, FX). Adapters group connections to a single communication link between two FB instances, which reduces *Diffuseness* of the application diagram (OK). However, the limited accessibility of the abstracted pins also reduces *Visibility* (FX). Compound data types ("Structs") could group data connections and increase *Abstraction*, but are not supported in 4diac IDE (FX).

2) Tool improvements: We added a feature to move FB instances from a subapp to the surrounding FB network while automatically adjusting the SubApp interface. A tabular editor was developed for creating Structs and we improved the data type selection. For all Structs and Adapters, we added a link to the type editor to quickly access the abstracted pins.

C. Working with types

Typed subapps are defined in IEC 61499 and are stored in the library like FB types. Additionally, 4diac IDE supports untyped subapps that are used only in a single location. Their functionality resembles anonymous classes in object-oriented languages. Users can save an untyped subapps as a type for later reuse, or detype a typed subapp to perform changes in a single instance (i.e., convert a typed subapp to an untyped subapp). They can also replace any instance with another type.

1) CD assessment: Detyping a subapp removes the connection to its type definition, thus turning it into a clone of the original type. Editing the type will not affect this instance anymore, leading to *Error Proneness*, as users may forget to also modify the untyped copy (ST). Graphically, untyped subapps can only be differentiated from typed subapps by their missing instance name (*Role Expressiveness*, FX). The interface of untyped subapps is shown in the Properties view, but not the one of typed subapps (*Consistency*, FX). If a pin names is renamed in a type, connections to this pin in its instances are lost, resulting in the need for numerous manual changes (*Viscosity*, ST). Furthermore, each FB instance has to be updated individually (*Viscosity*, FX).

2) Tool improvements: We implemented new features to automatically update or unmap all types in an editor to reduce the number of manual operations. FB instances now have an icon indicating their type. We also redesigned our icons and added a new icon for typed subapps to better differentiate them from untyped ones.

D. Editing

The 4diac IDE supports adding FB instances to the Application from the System Explorer and from the Palette. Both views are by default next to the graphical editor (cf. Fig. 2). Connections can be added via Drag&Drop between pins, if the data types are compatible.

1) CD assessment: The *Viscosity* for changing the layout of an application is very high, as all affected FB instances and connections have to be adjusted individually (FX). This high viscosity may increase the *Enforced Lookahead*, when developers have to know the target program structure early. For adding new FB instances, the respective type has to be selected outside the graphical editor, which may constitute a *Hard Mental Operation* (FX). Adding or changing connections is difficult due to the lack of visual feedback that illustrates which parts a user can interact with (*Visibility*, FX).

2) Tool improvements: We created a dedicated in-place field for searching types and inserting instances directly in the editor. It can be quickly accessed by double-clicking onto the diagram background. We added a selection and hover feedback to connections to improve their handling. Furthermore, we added support for the Eclipse Layouting Kernel [38] to allow for automated placement of FB instances and connections.

E. Cross-cutting Aspects

4diac IDE always enforces correct models, which impacts *Provisionality* and *Premature commitment*. We decided to study these dimensions in more detail in our user study (ST). To prepare for the study, we identified and fixed several general issues, especially regarding *Consistency* (FX). Specifically, we revised our menu entries to be consistent with those of the Eclipse platform. Where IEC 61499-specific terms are introduced (e.g., a System), they are now used consistently also within dialogues. Considering our graphical editors, we learned that the framework GEF3 [36] does not handle zooming nor scrolling correctly. This was a showstopper we had to fix (FX) for handling large applications.

TABLE I
MAIN RESULTS OF COGNITIVE ASSESSMENT OF 4DIAC IDE

Cognitive Dimension	Tool support	C.A.	User Study
Secondary Notation	names, comments	OK	+/-
Role Expressiveness	FB representation	ST	-
Abstraction	subapps, adapters	OK	-
	structs	FX	+
Visibility	abstracted contents	FX	-
Juxtaposability	arrange views	OK	+
Diffuseness	selection, hover	FX	+
	bidirectional link	OK	+
Hard Mental Operation	follow signal	FX	+/-
Viscosity	untyped subapp	OK	+
	type updates	ST	+/-
Enforced Lookahead	automated layout	FX	+/-
Error Proneness	refactoring support	OK	+
Premature Commitment	move blocks, layout	FX	+/-
Consistency	names, icons	FX	+/-
Provisionality	correct models	ST	+

VI. DESIGN OF THE USER STUDY

Based on four typical maintenance tasks (cf. Section II-A), the goal of our user study was to cover the key capabilities of IDEs for visual languages with respect to handling large and complex applications. We tested the task difficulty ourselves and in test runs with PhD students from our lab. Furthermore, we conducted a pilot study with two students from the computer science field, who have never used 4diac IDE before, and with an industrial expert. Based on their feedback, we made minor adaptations to the study method, e.g., we reduced ambiguities by rephrasing text in the instructions given to users before the study. To increase the participation of industrial users, our goal was to ensure that subjects could complete all tasks in less than one hour, while still covering the key

maintenance activities. We also considered the results of our initial cognitive dimensions marked as ST or FX in Section V.

A. Study System and Subjects

The study was conducted using an example application that followed a hierarchical design as described in guidelines from [39]. The application models the functionality of a capping station, where a robot places a lid (from a feeding station) on parts (arriving on a conveyor belt). An application for this mechatronic station has been previously published in [30]. We extended it to cover three identical parallel stations (referred to as Left-, Middle-, and RightCappingStation). Furthermore, we introduced additional hierarchies to better serve the purpose of the study.

Ten experienced automation engineers were nominated by our industry partner as subjects in the study. The subjects had an average of 14 years of experience in control software development, ranging from 1 year to 30 years. They have been working for their current employer between 4 and 30 years, on average 13.6 years. All subjects have an educational background in engineering, eight of them from a college or a university. One subject served as a pilot subject to ensure flawless operation of the study and to reveal problems in the setup. Seven subjects had participated in at least one workshop on developing control software with IEC 61499 and already had at least basic knowledge of software development based on this standard, as well as basic experience using the tool 4diac IDE. Three subjects used 4diac IDE for the first time during the study. To ensure basic knowledge of the language IEC 61499 and 4diac IDE, we created a video [40] (7 minutes long) where we outlined both. The video shows the version of 4diac IDE that was used in the study. We did not explain the features required in the study in detail to also assess the discoverability of features in a complex IDE such as 4diac IDE. Using a prerecorded video ensured that all subjects received the same information prior to the study.

B. Study Process and Data Collection

The study was conducted in a remote setting via a video conferencing tool (Zoom or Skype). If the subject agreed, the session was recorded including the screen with all mouse movements and the audio (recorded 9 sessions). We conducted the following process separately with each subject.

1) Briefing: The moderator first explained the goals and purpose of the study to the subject and requested their consent for participancy in the study. Also, the moderator asked whether the subject had watched the introduction video and whether there were any open questions. We asked the subject to activate the webcam, so that we could observe the subject, e.g., facial expressions, when performing the tasks. The subject had to share their screen in the call so that all their actions could be observed. As a last step, the moderator assisted the subject in starting 4diac IDE and importing the 4diac IDE project that contained the study system. The next phase of the study started as soon as the control application was opened in the Application editor of 4diac IDE.

2) Tasks: Each subject performed the tasks described in Section IV-C. We asked each subject to "think aloud" [11], i.e., to describe what s/he was doing and to comment on any concerns. One scribe documented the think-aloud statements. Another scribe watched the subject, who performed the maintenance tasks, and took additional notes on interesting observations beyond the think-aloud protocol.

3) Data Collection: After the subject had completed all tasks, the moderator performed semi-structured interviews on utility and usability with each subject, covering questions on the results of the cognitive dimensions assessment (cf. Section V). Regarding usability, we asked questions such as "How did you like the tool capabilities for restructuring the application" or "How did you like the possibility to view the contents of a subapp within its context?". Finally, the subject got a link to access a usability questionnaire that was created with the tool LimeSurvey and hosted on a server of our university. The questionnaire is based on Nielsen's usability attributes [12] and covers the five tool editors and views of 4diac IDE presented in Fig. 2. We phrased the attributes as questions, e.g., "How easy was it to learn working with 4diac IDE?". Regarding utility, we asked questions [31] such as "What opportunities do you see for your company when using this tool in daily business?". We also collected demographic information. The templates that we used for writing think-aloud and observer protocols, the usability questionnaire, the list of tasks, as well as the questions of the interview are available online [41].

C. Data Analysis and Reporting

All think-aloud protocols and observer notes were stored on a cloud server. Using an open coding technique [13], one researcher related all statements to the activities and tool capabilities. This work was checked by two other researchers. A total of over 900 think-aloud statements and 370 observations were recorded by the scribes. Per subject, we collected about 10 pages of material. In a joint session, all authors assigned the identified statements to the cognitive dimensions. We could directly relate many think-aloud statements with the cognitive dimensions discussed in Section V. We discussed the interpretation of all think-aloud statements and observer notes as well as the answers given by study subjects in the interviews to derive implications on usability and utility (Section VII) and also general implications for tool developers (Section VIII). As we related interview questions regarding usability with activities and cognitive dimensions, we can discuss the subjects' answers in the light of the CD framework.

VII. Study Results and Discussion

In our discussion, we report results from the ten industrial experts who participated in the study (including the subject who served as a pilot). We focus on results related to dimensions that required further analysis according to our cognitive assessment, i.e., that were marked as ST (we wanted to investigate these in more detail in the study) or FX (to evaluate the usefulness of our tool improvements). Quantitative results

of the questionnaire are reported in Table II, qualitative results are summarized in Table III. For each task, we present detailed results and relate each aspect to the cognitive dimensions (cf. Table I).

1) Orienting in an unknown application: All subjects relied on *Secondary Notations* to find the application parts that represent a motor. 2 subjects asked for adapting the concrete syntax of an FB to graphically represent its functionality (*Role Expressiveness*). 6 subjects wanted to use a search feature for finding an instance by its name (*Visibility*). After identifying the subapp type `Motor`, 5 subjects furthermore requested a direct link to all instances of a certain type (*Hidden Dependencies*). As 4diac IDE does not support an automated search, subjects had to navigate through the application manually, either via the tree view (System Explorer) or in the Application editor. The implemented shortcuts for navigation helped subjects to quickly move across hierarchy levels, yet we observed that a move from one level to another still poses a *Hard Mental Operation*. 4 subjects mentioned that the path to an FB instance is not *visible* in the editor, which may have facilitated orienting in the application. 6 subjects specifically reported difficulties in identifying the current editing location (*Hidden Dependency*). We could not observe any major difficulties in selecting FB instances or pins, unlike the situation in our cognitive assessment. Hence, the new selection feedback may have decreased *Diffuseness*. In 4diac IDE, each hierarchy level is opened in a separate editor tab, thus affecting Diffuseness. In the interview, we asked subjects whether they had preferred navigating within a single tab. 7 subjects considered it important to view application parts side-by-side, but 5 subjects prefer a single tab as default mechanism. We therefore recommend visualizing the full path to the contents of the editor tab. Concerning *Abstraction*, one subject positively remarked that the hierarchy structures the application and reduces the diagram size. 2 other subjects however mentioned that the deep hierarchy of the demo application hindered understanding the application. The possibility to expand a subapp aimed at increasing the *Visibility*, but only 2 subjects used it actively. During the interview, the feature was however considered useful by 5 subjects. Further enhancements may be required to improve the utility of this feature.

2) Creating/removing hierarchies: All subjects positively remarked the low *Viscosity* of editing untyped subapps. We furthermore did not observe any major issues while moving FB instances across the hierarchy (*Premature Commitment*). The subjects had to create a new subapp from three FB instances. We observed that 2 subjects first created a subapp and then added the FBs, while 8 subjects used the dedicated feature that creates a subapp directly from a selection of a set of FBs. 4diac IDE does not suggest a specific order of operations for this task (*Premature Commitment*). One subject criticized the automatically generated pin names. They are created as a combination of the FB instance name and the pin name, which may violate coding guidelines for a software project. As a result, each pin name would have to be adjusted manually (*Viscosity*). For deleting a hierarchy level, subjects used various approaches: 5 found the dedicated feature, 3 moved the contained FB instance and manually deleted the empty subapp, and 2 used cut&paste. The last approach does not automatically update the connections, which was requested as an improvement by 5 subjects throughout the study. As cut&paste stores connections within the same hierarchy level, this request can be related to the cognitive dimension *Consistency*. No subject expanded the subapp to increase *Visibility*, which would have allowed drag&drop of the contained FB. In the interview, we asked subjects whether they liked the possibilities for restructuring an application. 7 subjects confirmed that they liked the current refactoring features, but 5 of them requested further improvements.

3) Working with types: In this task, subjects had to save their subapp, which was created in the previous task, to the type library. Within the respective dialogue, 5 subjects struggled to select the right target type library from the list of all projects in the workspace. Hence, 2 of them requested that the type library of the currently edited project should be pre-selected by default (*Diffuseness*). 5 subjects had difficulties in creating the required folder in the type library. Specifically, they expected a dedicated button or context menu entry in the dialogue, while folders can only be added by specifying the new folder name as part of the save path, or before opening the dialogue (in the system explorer). We can relate this observation to the cognitive dimension *Premature Commitment*. We also observed difficulties in distinguishing the provided views: one subject tried to create the new folder in the Palette, which is only used for adding instances to the Application (*Consistency*).

Subjects also had to add a type file from the file system to their project library. This task can be completed via drag&drop or copy&paste from the file explorer to the type library. 9 subjects had difficulties in adding the file. One of them, who had no prior experience in 4diac IDE, could not complete this task without detailed instructions from the moderator (*Hard Mental Operation*). 9 subjects tried to use the Import dialogue, which, however, does not support importing single files (*Consistency*). Next, we asked subjects to replace their own motor controller with an instance of the newly imported type. The automatically generated pin names do not match those of the imported type and, therefore, 4diac IDE does not automatically handle the connections. As connections are dismissed without a prior warning (requested by one subject), this targets the cognitive dimension *Error Proneness*.

4) Editing: The typed subapp had to be detyped, which could be completed by all subjects. When instructed to add a data pin to the subapp interface, 5 subjects attempted to double-click on the background of the table to create a new row. As a double-click allows adding new FB instances in the graphical editor, this can be classified as *Inconsistency*. Then, subjects had to add three FB instances to their subapp. Most subjects used the in-place search field that was implemented to avoid a *Hard Mental Operation*. It is accessible via double-click, or via the context menu (preferred way of one subject).

TABLE II
RESULTS FROM THE QUESTIONNAIRE: USABILITY ASPECTS

	Learnability					Number of Errors					Subjective Satisfaction					Efficiency				
View	v.e.	e.	d.	v.d.	n/a	n.	s.	m.	t.m.	n/a	v.p.	p.	u.	v.u.	n/a	v.e.	e.	i.	v.i.	n/a
System Explorer	1	6	2	0	1	5	3	0	0	2	1	5	1	0	3	-	-	-	-	-
Application Editor	4	4	2	0	0	2	6	1	0	1	2	6	1	0	1	-	-	-	-	-
Properties View	2	5	3	0	0	2	6	0	0	2	1	5	2	0	2	-	-	-	-	-
Dialogues	0	6	1	0	3	5	0	1	0	4	0	5	1	0	4	-	-	-	-	-
Outline	5	1	0	0	4	4	1	0	0	5	2	2	0	0	6	-	-	-	-	-
Overall/4diac IDE	2.4	4.4	1.6	0	1.6	3.6	3.2	0.4	0	2.8	1.2	4.6	1	0	3.2	0	5	2	3	0

Learnability is rated from very easy (v.e.) to very difficult (v.d.), Number of errors from none (n.) to too many (t.m.), Subjective Satisfaction from very pleasant (v.p.) to very unpleasant (v.u.), and Efficiency of working with 4diac IDE is ranked from very efficient (v.e.) to very inefficient (v.i.).

TABLE III
QUESTIONNAIRE: UTILITY OF 4DIAC IDE

Strengths
Structuring Mechanisms (3) and dedicated Refactoring Features (3)
Automated Layouting (3)
Performance (2)
Working with Types (2) and Expanded Subapps (2)
Weaknesses
Appearance: Line Routing (5) and Layout of Expanded SubApp (2)
Search Features (3), Navigation in the System Explorer (2)
Graphical Representation of FBs (2) and Icons (2)
Opportunities
Platform-independent software (4)
Better meet requirements for development (4)
Threats
Currently not all required features are supported (4)
Small development community (3)
Long-term support (3) and Liability (2)
Customers request specific hardware (3)

One subject however preferred the Palette for creating new instances, but perceived its behavior as inconsistent with the System Explorer. Another subject used drag&drop from the System Explorer. 4 subjects mentioned that they liked the automated layout for applications. Whereas one subject requested that the tool automatically applies a new layout after adding a block (*Viscosity*), another subject preferred manually triggering this process. For 2 subjects, we observed difficulties in orienting in the application after applying a layout.

Although we improved handling and creating connections, we observed difficulties in performing this task. For instance, reconnecting requires first selecting a connection, but subjects attempted to immediately drag the handle. As moving FB instances is possible without prior selection, this was regarded as an *Inconsistency* by 3 subjects. Also, for one subject, the weight of the connections was too small (*Visibility*). 6 subjects furthermore requested that the routing for newly created connections must be improved (*Viscosity*).

We asked subjects to identify which variables are contained in the Struct named `ctrl`. A quick link for accessing its contents is available in many locations of the Properties view. As a result, all subjects found the type definition quickly, indicating a benefit of the implemented redundancy (*Visibility*). We also requested that subjects enter constant values for parameters. One subject expected better consistency between the Properties view and the graphical editor. Edits should be projected immediately, not only after the new entry has been confirmed (*Visibility*). 4 subjects furthermore considered the input validation as insufficient (*Error Proneness*).

5) Cross-cutting Aspects: We further identified issues that are relevant for all of our tasks. Subjects had difficulties with the naming of some context menu entries, especially where they considered several menu entries as potentially fitting for their current task (*Role Expressiveness*): for instance, `Flatten subapp` removes a hierarchy level and replaces a subapp with its contents, while `Toggle SubApp Representation` expands a subapp (2 subjects) The icon size was reported to be too small (2 subjects, *Visibility*). 4 subjects had difficulties understanding the icons and 2 could not differentiate them (*Consistency*).

VIII. LESSONS LEARNED

We summarize five lessons that we learned from our study and that we consider relevant for developers of any graphical editor for visual modeling or programming languages.

Beauty is in the eye of the beholder: manually arranging graphical diagrams is tedious. Like whitespace in textual languages, developers use the two-dimensional arrangement of blocks in visual languages to convey information (secondary notation) [42]. The memorability of software parts is also important for quickly navigating through the model. As a result, several users requested an additional layout algorithm that does not alter the block position. The requirements for the layout vary depending on the user and the software and should therefore be customizable in the tool. For instance, some users may need to specify long parameter values and require more space between blocks, whereas others may prefer a compact representation to get a better overview of their diagram.

Lost&Found: efficient search features are essential. Users benefit from advanced search capabilities when orienting in unknown diagrams. Search boxes should always be provided as an alternative to trees, drop-down menus, and lists to develop truly scalable IDEs. For instance, most subjects requested features for searching instances by their name. Furthermore, searches can reveal relations that are otherwise not directly accessible: in our context, users requested an overview of the used instances of a type, like the call hierarchy for functions that is common in textual editors.

One at a time: offer one view per task. Separation of concerns is essential to manage complex software [43]. Subjects considered the System Explorer difficult to use, mainly because it is used as both a file explorer and for type

management, while additionally showing the application structure. Furthermore, configurable perspectives as offered by the Eclipse platform could be used to customize the System Explorer and other views to better align them with the complex development tasks specific to the respective engineering roles.

Living with inconsistencies: handle incorrect models gracefully. Input validation should ensure correct and consistent models. However, a too strong focus on correctness may enforce a very strict order of user interactions, especially early in the development process [44]. Where errors cannot be avoided, users should be informed unobtrusively, but near the editing position. Consider that users may be required to interrupt their work and continue later. For this use case, also incorrect models have to be visualized with best effort. Dedicated features should support users to return step-by-step to a correct and consistent model.

Get a new perspective: multi-stage usability studies are worth the effort. The cognitive dimensions approach and the user study allowed us to improve the usability of our tool. The applied multi-stage process allowed us to receive feedback from several perspectives. Issues were reported in diverse stages. Some aspects were observed during the practical tasks, discussed in the interview, and reported later again by the subject in the questionnaire. Other aspects only appeared in the questionnaire, which the subjects could complete in their own pace, and which therefore complemented our observations well. Not all issues would have been revealed if the study had only comprised a single stage.

IX. Threats to Validity

A threat to construct validity is the potential bias caused by the system created for the study. Although the control application is based on prior publications [30], [39], the final system was created specifically for the study by one of the authors. However, our study does not focus on model details, but utilizes the model to evaluate the usefulness of the tool. We selected the model of the capping station as it was sufficiently large and it was expected to be intuitively understood by the domain experts, yet sufficiently different from their daily business to evaluate the tool rather than the model.

There are also threats to internal validity meaning that the results might have been influenced by our treatment. We had no direct influence on the selection of subjects, instead, they were nominated by our industry partner based on our requirements. We could therefore not ensure that the subjects represent a variety of departments. The number of subjects (ten) may seem relatively small. However, they cover a range of different roles and a very wide range of work experience in the domain. Their average experience in developing control software is extensive (14 years). Furthermore, relatively few subjects can reveal a high percentage of the total usability issues [45].

Regarding conclusion validity, there is a threat that the results are not based on statistical relationships or measurements but on qualitative data [13]. Given that the main aim of the study was to investigate the behavior and opinions of users of a tool, qualitative research methods are though well suited. The analysis of the collected data still depends on our interpretation. The work was performed by a single researcher but the result was carefully checked by two senior researchers.

With respect to external validity, we selected an example system that is representative regarding the size and complexity for the control software domain. Although the derived implications depend on our experiences of using and implementing 4diac IDE and especially its GUI, the capabilities are common in other IDEs for visual modeling languages, as discussed in Section II-B. The mapping to the CD framework relates our results to HCI knowledge.

Our results are specific for our setting and our example system but can be generalized to some degree, as our tool is similar to those in the same domain. It is furthermore based on open source infrastructure, which is frequently used by other modeling tools. Identified issues are thus potentially applicable to other Eclipse-based tools too.

X. Conclusions

Industrial experts in control software handle large-scale applications for production lines and have to consider the long life cycle of the mechanical components, which also motivates frequent control software maintenance. In this paper, we assessed the usefulness, i.e., usability and utility, of 4diac IDE, a visual programming IDE for large-scale automation software, for common control software maintenance tasks. We first performed an initial assessment of the tasks in a walkthrough of the IDE following an approach based on the Cognitive Dimensions of notations framework and fixed usability flaws. In a user study, ten industrial experts then evaluated these tasks. Our results and lessons learnt from the study are relevant for developers of visual modeling and programming tools. Identified capabilities are often not sufficiently supported in such tools. Furthermore, our results with an Eclipse-based tool are potentially applicable to other modeling tools using the same technology. As our improvements are available as part of the latest open-source release of the 4diac IDE, they can serve as a good practice example for other project teams, together with this paper. Our results and lessons learnt complement existing results and findings by adding experiences made in the domain of industrial automation to the body of knowledge of developing visual modeling tools. In general, we conclude that advanced tools with advanced editing support can simplify working with large models and thus increase the benefits of the applied modeling language. In future work, we will further investigate dimensions that we identified as insufficiently addressed by the tool. Hence, we will study both, adaptions to the language IEC 61499 as well as improvements for IDEs.

XI. Acknowledgments

The financial support by the Austrian Federal Ministry for Digital and Economic Affairs, the National Foundation for Research, Technology and Development, and the Christian Doppler Research Association is gratefully acknowledged. We particularly thank all participants of the study.

References

[1] R. Jolak, M. Savary-Leblanc, M. Dalibor, A. Wortmann, R. Hebig, J. Vincur, I. Polasek, X. Le Pallec, S. Gérard, and M. R. V. Chaudron, "Software engineering whispers: The effect of textual vs. graphical software design descriptions on software design communication," *Empirical Software Engineering*, vol. 25, no. 6, pp. 4427–4471, 2020.

[2] S. Meliá, C. Cachero, J. M. Hermida, and E. Aparicio, "Comparison of a textual versus a graphical notation for the maintainability of MDE domain models: an empirical pilot study," *Software Quality Journal*, vol. 24, no. 3, pp. 709–735, 2016.

[3] National Instruments, "Labview 2020," 2020, Accessed: May 12, 2021. [Online]. Available: www.ni.com/en-us/shop/labview.html

[4] The MathWorks, "Simulink R2021a," 2021, Accessed: May 12, 2021. [Online]. Available: www.mathworks.com/products/simulink.html

[5] International Electrotechnical Commission (IEC), "IEC 61131 - Programmable controllers, Part 3: Programming languages," Geneva, 2014.

[6] International Electrotechnical Commission (IEC), TC65/WG6, "IEC 61499-1, Function Blocks - part 1: Architecture: Edition 2.0," Geneva, 2012.

[7] R. Harrison, D. Vera, and B. Ahmad, "Engineering Methods and Tools for Cyber–Physical Automation Systems," *Proceedings of the IEEE*, vol. 104, no. 5, pp. 973–985, 2016.

[8] B. Vogel-Heuser, A. Fay, I. Schaefer, and M. Tichy, "Evolution of software in automated production systems: Challenges and research directions," *Journal of Systems and Software*, vol. 110, pp. 54–84, 2015.

[9] B. Vogel-Heuser, C. Diedrich, A. Fay, S. Jeschke, S. Kowalewski, M. Wollschlaeger, and P. Göhner, "Challenges for Software Engineering in Automation," *Journal of Software Engineering and Applications*, vol. 7, no. 5, pp. 440–451, 2014.

[10] M. Törngren and P. Grogan, "How to Deal with the Complexity of Future Cyber-Physical Systems?" *Designs*, vol. 2, no. 4, p. 40, 2018.

[11] A. Holzinger, "Usability engineering methods for software developers," *Communications of the ACM*, vol. 48, no. 1, pp. 71–74, 2005.

[12] J. Nielsen, *Usability engineering*. Morgan Kaufmann, 1994.

[13] C. B. Seaman, "Qualitative methods in empirical studies of software engineering," *IEEE Transactions on Software Engineering*, vol. 25, no. 4, pp. 557–572, 1999.

[14] R. Holwerda and F. Hermans, "A usability analysis of blocks-based programming editors using cognitive dimensions," in *2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 2018, pp. 217–225.

[15] T. Weber, A. Zoitl, and H. Hußmann, "Usability of development tools: A case-study," in *2019 ACM/IEEE 22nd Int. Conf. on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. IEEE, 2019, pp. 228–235.

[16] M. Obermeier, S. Braun, and B. Vogel-Heuser, "A Model-Driven Approach on Object-Oriented PLC Programming for Manufacturing Systems with Regard to Usability," *IEEE Transactions on Industrial Informatics*, vol. 11, no. 3, pp. 790–800, 2015.

[17] G. Bayrak, F. Ocker, and B. Vogel-Heuser, "Evaluation of Selected Control Programming Languages for Process Engineers by Means of Cognitive Effectiveness and Dimensions," *Journal of Software Engineering and Applications*, vol. 10, no. 5, pp. 457–481, 2017.

[18] R. Rabiser, M. Vierhauser, and P. Grünbacher, "Assessing the usefulness of a requirements monitoring tool: A study involving industrial software engineers," in *2016 IEEE/ACM 38th Int. Conf. on Software Engineering Companion (ICSE-C)*, 2016, pp. 122–131.

[19] Eclipse 4diac. (2020) Eclipse 4diac - The Open Source Environment for Distributed Industrial Automation and Control Systems. Accessed: May 12, 2021. [Online]. Available: www.eclipse.org/4diac

[20] A. Blackwell and T. Green, "Notational Systems—The Cognitive Dimensions of Notations Framework," in *HCI Models, Theories, and Frameworks*. Elsevier, 2003, pp. 103–133.

[21] B. Vogel-Heuser, S. Feldmann, J. Folmer, J. Ladiges, A. Fay, S. Lity, M. Tichy, M. Kowal, I. Schaefer, C. Haubeck, W. Lamersdorf, T. Kehrer, S. Getir, M. Ulbrich, V. Klebanov, and B. Beckert, "Selected challenges of software evolution for automated production systems," in *2015 IEEE 13th Int. Conf. on Industrial Informatics (INDIN)*, 2015, pp. 314–321.

[22] M. Obermeier, S. Braun, K. Sommer, and B. Vogel-Heuser, "Fundamental Aspects Concerning the Usability Evaluation of Model-Driven Object Oriented Programming Approaches in Machine and Plant Automation," in *Design, User Experience, and Usability. Theory, Methods, Tools and Practice*, ser. Lecture Notes in Computer Science, A. Marcus, Ed. Springer Berlin Heidelberg, 2011, vol. 6770, pp. 497–506.

[23] C. Legat, J. Folmer, and B. Vogel-Heuser, "Evolution in industrial plant automation: A case study," in *IECON 2013 - 39th Ann. Conf. of the IEEE Industrial Electronics Society*. IEEE, 2013, pp. 4386–4391.

[24] J. Fischer, B. Vogel-Heuser, F. Haben, and I. Schaefer, "Reengineering workflow for planned reuse of IEC 61131-3 legacy software," in *2020 IEEE Int. Conf. on Industrial Engineering and Engineering Management (IEEM)*. IEEE, 2020, pp. 1126–1130.

[25] A. Hopsu, U. D. Atmojo, and V. Vyatkin, "On Portability of IEC 61499 Compliant Structures and Systems," in *2019 IEEE 28th Int. Symp. on Industrial Electronics (ISIE)*. IEEE, 2019, pp. 1306–1311.

[26] G. Lyu and R. W. Brennan, "Towards IEC 61499-Based Distributed Intelligent Automation: A Literature Review," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 4, pp. 2295–2306, 2021.

[27] nxtControl GmbH, "Nxt Technology IDE," Accessed: May 12, 2021. [Online]. Available: www.nxtcontrol.com/engineering/

[28] Rockwell Automation, "ISaGRAF Development Toolkit," Accessed: May 12, 2021. [Online]. Available: www.isagraf.com

[29] Holobloc, Inc., "Function Block Development Kit (FBDK)," 2020, Accessed: May 12, 2021. [Online]. Available: www.holobloc.com

[30] A. Zoitl, T. Strasser, and G. Ebenhofer, "Developing modular reusable IEC 61499 control applications with 4DIAC," in *2013 11th IEEE International Conference on Industrial Informatics (INDIN)*. IEEE, 2013, pp. 358–363.

[31] F. D. Davis, "Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology," *MIS Quarterly*, vol. 13, no. 3, p. 319, 1989.

[32] R. Rabiser, P. Grünbacher, and M. Lehofer, "A qualitative study on user guidance capabilities in product configuration tools," in *2012 Proc. 27th IEEE/ACM Int. Conf. on Automated Software Engineering*, M. Goedicke, T. Menzies, and M. Saeki, Eds. ACM Press, 2012, p. 110.

[33] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Software Engineering*, vol. 14, no. 2, pp. 131–164, 2009.

[34] A. J. Ko, T. D. LaToza, and M. M. Burnett, "A practical guide to controlled experiments of software engineering tools with human participants," *Empirical Software Engineering*, vol. 20, no. 1, pp. 110–141, 2015.

[35] A. Zoitl and H. Prähofer, "Guidelines and Patterns for Building Hierarchical Automation Solutions in the IEC 61499 Modeling Language," *IEEE Transactions on Industrial Informatics*, vol. 9, no. 4, pp. 2387–2396, 2013.

[36] D. Rubel, J. Wren, and E. Clayberg, *The eclipse graphical editing framework (GEF)*, ser. The eclipse series. Addison-Wesley, 2012.

[37] Eclipse 4diac. (2021) Maintaining Control Software in 4diac IDE: Youtube. [Online]. Available: https://youtu.be/xishphcgYmc

[38] Eclipse Layouting Kernel. (2020) Eclipse Layout Kernel 0.6.1. Accessed: May 12, 2021. [Online]. Available: www.eclipse.org/elk/

[39] A. Zoitl and R. W. Lewis, *Modelling control systems using IEC 61499*, 2nd ed., ser. IET Control engineering series. London: IET, 2014, vol. 95.

[40] Eclipse 4diac. (2021) Overview of IEC 61499 and Eclipse 4diac: Youtube. [Online]. Available: https://youtu.be/K9iItQBC-ac

[41] B. Wiesmayr, A. Zoitl, and R. Rabiser. (2021) Assessing the Usefulness of a Visual Programming IDE for Large-Scale Automation Software: Documents for performing a user study on maintaining hierarchical control applications with 4diac IDE. Zenodo. [Online]. Available: doi.org/10.5281/ZENODO.4758816

[42] H. Fuhrmann and R. von Hanxleden, "Taming Graphical Modeling," in *Model driven engineering languages and systems*, ser. Lecture Notes in Computer Science, D. C. Petriu, N. Rouquette, and Ø. Haugen, Eds. Springer, 2010, vol. 6394, pp. 196–210.

[43] A. Cicchetti, F. Ciccozzi, and A. Pierantonio, "Multi-view approaches for software and system modelling: a systematic literature review," *Software & Systems Modeling*, vol. 18, no. 6, pp. 3207–3233, 2019.

[44] R. Balzer, "Tolerating inconsistency (software development)," in *1991 Proc. 13th Int. Conf. on Software Engineering*. IEEE, 1991, pp. 158–165.

[45] J. Nielsen and T. K. Landauer, "A mathematical model of the finding of usability problems," in *Proc. INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems*, ser. CHI '93. New York, NY, USA: Association for Computing Machinery, 1993, p. 206–213.