

**1      Title 3**

**2      ANNE-MARIE ROMMERDAHL, SDU, Denmark**

**3      JEREMY ALEXANDER RAMÍREZ GALEOTTI, SDU, Denmark**

**4      DIMITRIOS DAFNIS, SDU, Denmark**

**5      NASIFA AKTER, SDU, Denmark**

**6      MOHAMMAD HOSEIN KARDOUNI, SDU, Denmark**

**7      CCS Concepts:** • **Do Not Use This Code → Generate the Correct Terms for Your Paper;** *Generate the Correct Terms for Your Paper;* **Generate the Correct Terms for Your Paper;** **Generate the Correct Terms for Your Paper.**

**8      Additional Key Words and Phrases:** Do, Not, Use, This, Code, Put, the, Correct, Terms, for, Your, Paper

**9      ACM Reference Format:**

**10     Anne-Marie Rommerdahl, Jeremy Alexander Ramírez Galeotti, Dimitrios Dafnis, Nasifa Akter, and Mohammad Hosein Kardouni. 2018.  
11     Title 3. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation email (Conference acronym 'XX)*.  
12     ACM, New York, NY, USA, 16 pages. <https://doi.org/XXXXXXX.XXXXXXX>**

**13     1 Introduction**

This paper addresses these three questions:

- RQ1: How does OpenRoberta with the Reuse Assistant compare to the standard OpenRoberta in terms of task performance?
- RQ2: How well does the Reuse Assistant perform in terms of usability?
- RQ3: To what effect does the Reuse Assistant promote the creation and usage of reusable custom blocks?
  - Which factors influenced the end user's decision on whether or not to make use of the Reuse Assistant?

**14     2 Background and Related Work**

Software reuse is a broad term, that refers to the practice of reusing previously written code, rather than coding from scratch. It is such an important part of software engineering, that one of the ways to measure the quality of software is by its 'Reusability'[2], i.e. the degree to which the application or its components can be reused. There are multiple benefits to practicing reuse in software engineering. One developer could save time by using another developer's reusable component, rather than coding their own. The developer avoids both the work of writing the syntax and designing the logic of the component. The developer can design their own reusable components, keeping all the logic

---

Authors' Contact Information: Anne-Marie Rommerdahl, SDU, Odense, Denmark, anrom25@student.sdu.dk; Jeremy Alexander Ramírez Galeotti, SDU, Odense, Denmark, jeram25@student.sdu.dk; Dimitrios Dafnis, SDU, Odense, Denmark, didaf25@student.sdu.dk; Nasifa Akter, SDU, Copenhagen, Denmark, naakt23@student.sdu.dk; Mohammad Hosein Kardouni, SDU, Odense, Denmark, mokar25@student.sdu.dk.

---

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2018 Copyright held by the owner/author(s). Publication rights licensed to ACM.

Manuscript submitted to ACM

Manuscript submitted to ACM

in one place, which can then be tested thoroughly. However, despite reuse being an important practice in software engineering, there is still a limited focus on this practice when it comes to low-code development platforms (LCDP).

A study from 2021 studied several low-code platforms (LCPs), in order to identify characteristic features of LCPs. The identified features were presented according to how frequent they occurred, with domain-specific reference artifacts being categorized as 'rare'. Most studied systems offered catalogs of "reusable functions or examples of predefined processes", but they were found to be generic, or have a limited scope[3]. This lack of focus on promoting reuse may impact the so-called 'Citizen Developers', who have little or no coding knowledge, and whom may then miss out on the benefits of reuse. Lin and Weintrop (2021) noted that most existing research on block-based programming focuses on supporting the transition to text-based languages rather than exploring how features within BBP environments [4]—such as abstraction or reuse—can enhance learning outcomes.

There have been proposed some ideas on how to promote reuse for LCPs, such as the templating language OSTRICH, developed for model-driven low-code platform OutSystems[5]. OSTRICH is designed to assist the end-user in making use of OutSystems' available templates, by abstracting and parameterizing the templates. However, OSTRICH only supports the top nine most used production-ready screen templates, and does not allow the end-user to create and save their own templates, or re-apply a template which they have customized. Another approach focused on enabling the reuse of models, by providing recommendations to the end-user, based on the models stored in a graph acting as a repository. While the graph allows end-users to reuse their own models, there is no mention of guiding the user towards reusing their own models.

Several popular low-code development platforms (LCDPs) provide different kinds of support for reuse. Webflow[6], a LCDP for responsive websites, offers the ability to create reusable components and UI kits, which can be reused across multiple pages and projects. Mendix[7] and OutSystems offer even more functionality to support reuse, offering several ways to end-users to share their code with each other, and offering pre-made components. Both of these platforms also utilize AI to enhance reuse. Outsystems provides AI suggestions to spot and create reusable pieces, while Mendix uses AI to suggest the best solutions and components for specific tasks. However, for both of these platforms, the AI suggestions provided are not always accurate to successfully guide the end-user to create custom reusable components \*\*\* (How do we know this? What makes it 'accurate'?\*\*).

In order to analyze how block-based robotics environments address reuse, 4 representative platforms were compared: mBlock, MakeCode, SPIKE LEGO, VEXcode GO and Open Roberta. The comparison focused on three main dimensions of reuse: structural reuse (through user-defined blocks or functions), social reuse (through sharing or remixing existing projects), and interoperable reuse (through import/export capabilities).

Table 1. Block Based Robotics Environments Reuse Support

Platform	Structural Reuse	Social Reuse	Interoperable Reuse	Reuse Support
VEXcode GO	X	X		Medium
mBlock	X	X	X	Medium
MakeCode	X	X	X	Medium
Spike Lego	X		X	Low
Open Roberta		X		Low

In this context, “reuse support” represents a scale that measures how effectively each platform facilitates reuse-related features. High reuse support indicates that users can easily create, share, and adapt existing components or projects. Medium reuse support suggests that some reuse mechanisms are available but limited in scope or flexibility. Low reuse support implies that the platform provides only minimal or restricted features to promote reuse.

As shown in Table 1, although these platforms include reusability features, they are quite limited, as none of them provide users with clear guidance on how to use these tools effectively, which restricts their ability to fully leverage them.

A study by Techapalokul and Tilevich (2019) suggests that supporting mechanisms for reusing smaller, modular pieces of code can enhance programmer productivity, creativity and learning outcomes. Adler et al. (2021) introduced a search-based refactoring approach to improve the readability of Scratch programs by automatically applying small code transformations, such as simplifying control structures and splitting long scripts. Their findings demonstrated that automated refactoring can significantly enhance code quality and readability for novice programmers. Building upon this concept, our project applies similar principles in the OpenRoberta environment, focusing on detecting duplicate code segments and guiding users toward creating reusable custom blocks to promote modularity and abstraction.[1].

Existing block-based environments provide mechanisms for reuse, but lack intelligent support to help users recognize and apply reuse in practice. To address this gap, our project introduces a guided reuse assistant within the Open Roberta Lab environment. The tool is designed to help users identify and apply reuse more easily while creating their robot programs. It works by automatically scanning a user’s block-based program to detect repeated code segments in the workspace. The system visually highlights the found duplicates, drawing the user’s attention to patterns that could be simplified.

The tool also offers the functionality to create the custom block for the end-user, by identifying the small differences between the repeated parts—such as numbers, variables, or parameters—and turning these differences into inputs for the new block. The tool automatically replaces all relevant duplicate sequences with the new custom block.

By combining ideas from procedural abstraction (organizing code into meaningful, reusable parts) and automated refactoring (improving code through intelligent transformations), our tool aims to make block-based programming more structured and efficient. It encourages users to build programs that are modular and easier to maintain, helps reduce unnecessary repetition, and supports learning by making the concept of reuse clear and hands-on.

### 3 Study Design

Following the Design Science methodology, our study is structured into three main phases: problem investigation to define goals, treatment design to specify the artifact requirements, and treatment validation to assess the artifact’s performance in a controlled environment.

#### 3.1 Problem Investigation

*3.1.1 Problem Context and Motivation.* End-user development (EUD) for collaborative robots (cobots) presents unique challenges, particularly for users without formal programming training. In domains such as chemistry laboratories, educational robotics, and industrial settings, end-users need to program robots to perform specific tasks but often lack the software engineering knowledge to write maintainable, well-structured code. In the domain of Chemistry, one of the most relevant and important tasks is performing experiments in labs in order to test a hypothesis, or to aid in the understanding of how chemicals react. Robots can be used in chemistry labs to automate experiments with great effect, as many experiments involve steps that are repetitive, and susceptible to human error, such as a step being overlooked,

instructions being misread, etc. Automation of menial tasks will leave the chemists with more time for other work, and also comes with the added bonus of chemists not having to handle dangerous chemicals.

One critical challenge in EUD is code reuse. According to Blackwell's Attention Investment Model [cite that], users decisions about whether to engage in programming activites, are based on an implicit cost-benefit analysis. How big an investment must be made to create the program, how great will the pay-off be, and what is the risk that the investment won't pay off. While the usage of reusable components can be a great way to lower the cost of creating a program, the decision whether to create such a component is also subject to a cost-benefit analysis. Several visual programming environments, like OpenRoberta Lab, don't provide assistance in identifying when code should be reused or how to extract repeated sequences into reusable components. This leaves end-users with the added cost of having to figure out how each platform supports reuse, and how to make use of the provided features. As lab work in chemistry involves many repetitive tasks, these challenges can easily become an obstacle for the chemists, which may turn them away from using cobots, as the inconvenience outweighs the benefits.

**3.1.2 Stakeholder Analysis.** Chemists and lab technicians who use cobots for repetitive tasks such as sample preparation, dispensing, mixing, and quality control procedures. They possess deep domain expertise in chemistry but limited programming knowledge, often creating long, repetitive programs that become difficult to maintain when adapting experimental protocols. Their primary need is to quickly create and modify robot programs without becoming programming experts.

## 3.2 Treatment Design

To address the problem of code reuse in EUD for cobots, we have derived a set of requirements designed to contribute to the chemist's goal of creating maintainable and reusable robot programs. Functionally, the artifact must be capable of automatically detecting duplicate or similar block sequences and visually highlighting these duplications within the user's workspace. These requirements are necessary to help the end-user recognize opportunities for reuse, that would otherwise go unnoticed. Once detected, the system must suggest the creation of reusable custom blocks, allowing the user to accept or reject these suggestions. These signals are important, as they give the end-user control over the reuse process, allowing them to decide when and how to apply reuse in their programs. Regarding non-functional requirements, the artifact must seamlessly integrate with the existing Open Roberta Lab environment to ensure a smooth user experience. The interface should be intuitive for end-users, minimizing the learning curve and making it easy to understand and use the reuse features. Additionally, the artifact should not interfere with the existing workflow, allowing users to continue their programming tasks without disruption. Finally, clear visual feedback during the detection process is essential to help users understand what the system is doing and how to respond to its suggestions.

**3.2.1 Artifact Specification: The Reuse Assistant.** To satisfy the requirements above, we designed the Reuse Assistant as an extension of Open Roberta Lab.

**3.2.2 Architecture.** The system enables the execution of block-based programs on a simulated cobot through a three-tier architecture, as illustrated in 1. The workflow consists of the following stages:

- (1) **Client Side (Open Roberta):** The user interacts with the Open Roberta UI to assemble block sequences. The Reuse Assistant operates at this layer, analyzing blocks in real-time. Upon execution, the client generates specific data structures ("Generated Headers") representing the program logic.

- 209 (2) **Backend (Flask Server):** The client transmits these headers via HTTP POST requests to a Flask-based API  
 210 Endpoint. A "Translator" component processes the data, mapping the abstract block definitions to concrete  
 211 Python methods compatible with the robot's control logic.  
 212 (3) **Simulation (Mujoco):** The mapped methods trigger the execution of commands within the Mujoco Simulator,  
 213 which renders the physical behavior of the cobot in the virtual environment.

216 3.2.3 *Detection Algorithm.* The approach is intentionally simple so it is easy to read and to implement in a real block  
 217 editor. The algorithm follows three main steps:

- 219 • **Linearization:** First, the algorithm linearizes the block workspace into a sequential list of blocks.
- 220 • **Identify sequences:** It then iterates through this list to identify all possible sequences of blocks that meet a  
 221 minimum unique block type length requirement (three blocks) that can be repeated more than once.
- 222 • **Sequences Matching:** If the same sequence of block types is found more than once, it will be added to the  
 223 CustomReusableCandidates list which will eventually be sorted by longest and most recent duplicated sequences.  
 224 In the end the highest priority candidate gets returned.

226 The pseudocode below is short, explicit, and uses straightforward data structures (lists).

---

#### 229 **Algorithm 1** Duplicate Sequence Detection

230 **Require:** Workspace, StartBlock // user's block workspace  
 231 **Require:** MinimumSequenceLength = 3, MinimumDifferentBlockTypesInSequence = 3, MaxSequenceLength = 10  
 232 **Ensure:** ReusableComponentCandidates // list of repeated block sequences to return

```

 234 1: Chain = buildLinearChain(StartBlock)
 235 2: Sequences = List<sequence>
 236 3: for startIndex = 0 to length(Chain) - 1 do
 237   4:   for sequenceLength = 1 to MaxSequenceLength do
 238     5:     sequence = Chain[startIndex .. startIndex + sequenceLength - 1]
 239     6:     numberofBlockTypesInSequence = getNumberOfDistinctBlockTypes(sequence)
 240     7:     if sequenceLength >= MinimumSequenceLength and numberofBlockTypesInSequence >= MinimumDiffer-
 241       entBlockTypesInSequence then
 242         8:           Sequences.append(sequence) // record sequence occurrence
 243         9:         end if
 244       10:      end for
 245     11:    end for
 246   12:  ReusableComponentCandidates = {Sequences | occurrence ≥ 2}
 247   13:  sort ReusableComponentCandidates by (longest sequence length and most recent occurrence)
 248   14: return ReusableComponentCandidates[0] // Return highest priority candidate

```

---

254 Algorithm 1. Illustrates the core logic for identifying duplicate block sequences

256 3.2.4 *User Interface and Interaction.* The user interface is designed to be intuitive and non-disruptive. When the  
 257 detection algorithm identifies a candidate, the system visually highlights the blocks on the canvas as illustrated in  
 258 Figure 2. A non-blocking toast notification appears, prompting the user to confirm the refactoring. If confirmed, the  
 259

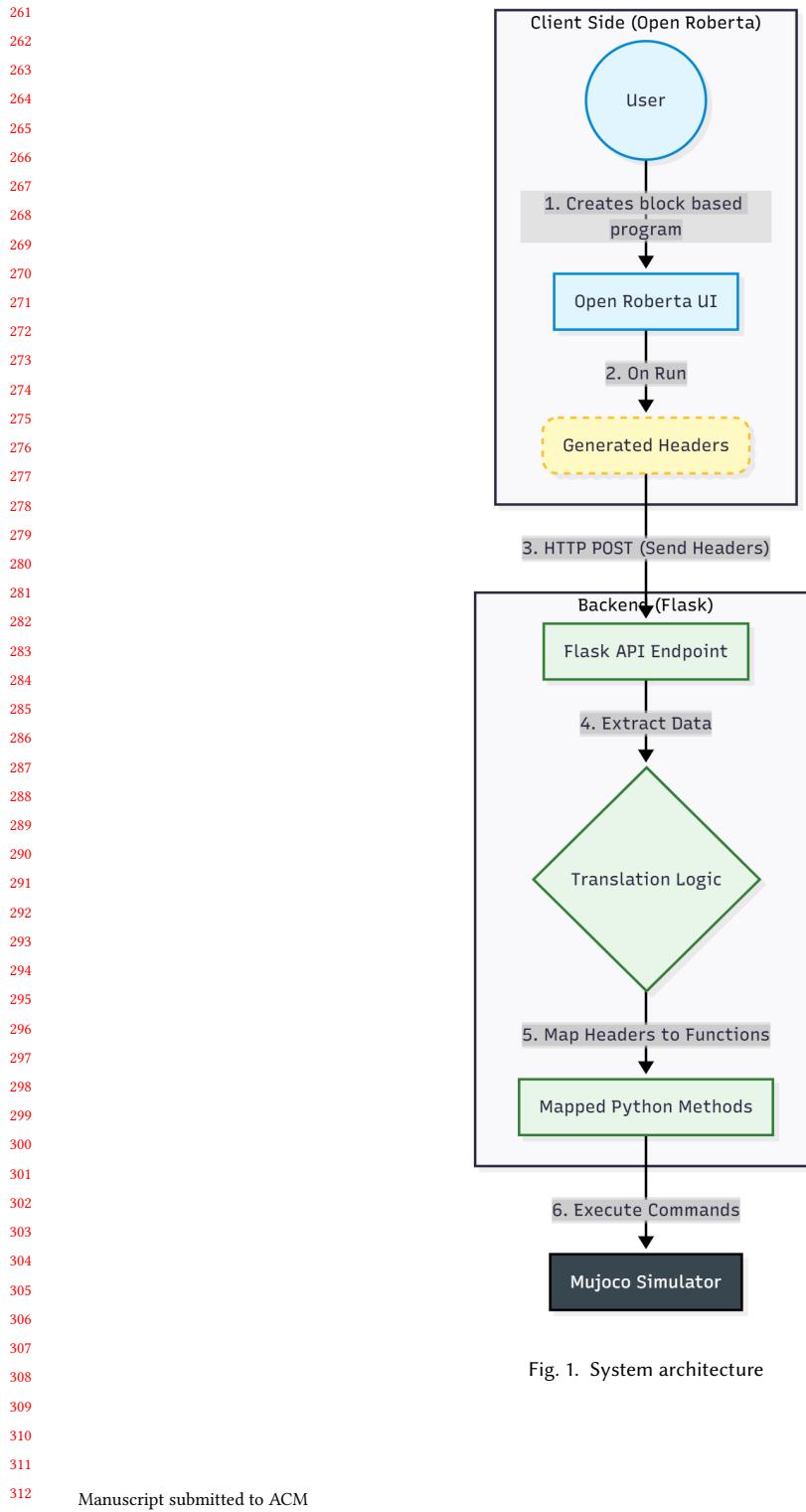


Fig. 1. System architecture

313 system automatically generates the custom block definition in a dedicated workspace area (handling visibility via  
 314 `revealDefinitionWorkspacePane`) and updates the main workspace, replacing the redundant code with concise  
 315 function calls as shown in Figure 3. This process abstracts the complexity of manual function creation, guiding the user  
 316 toward modular design practices. After the user presses the run simulation button, the robot simulator of mujoco opens  
 317 up and executes the commands provided by the user inside the Open Roberta workspace. This is illustrated in Figure 4.  
 318

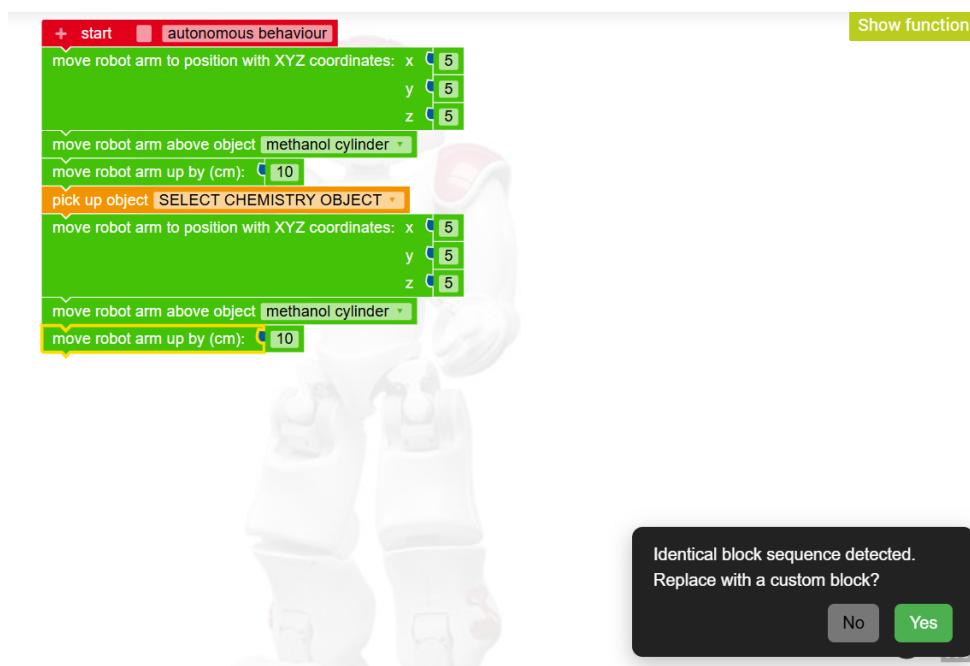


Fig. 2. Reuse Assistant workflow — detection: the interface detects and highlights duplicate blocks by changing their color to green.

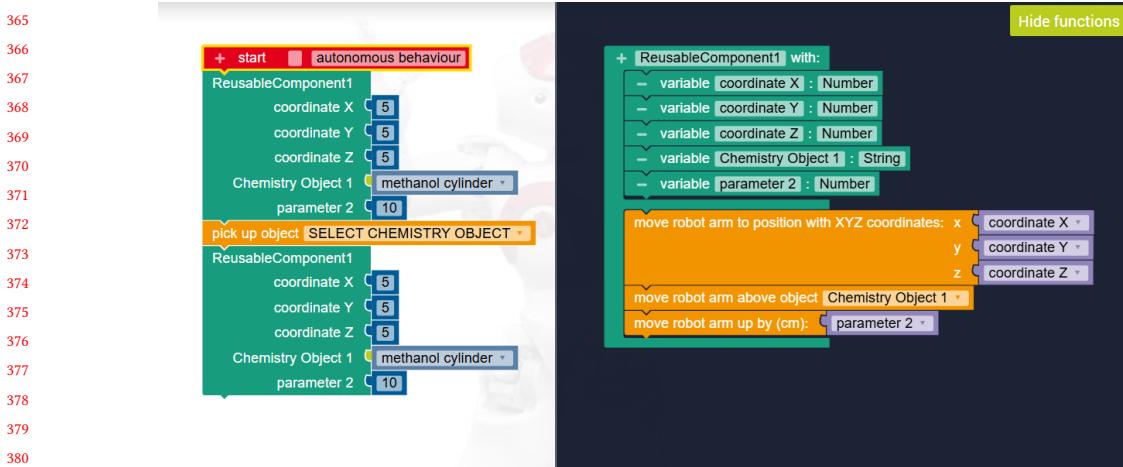


Fig. 3. Reuse Assistant workflow – refactoring: the automated refactoring result, showing the new custom block definition and the simplified main program.

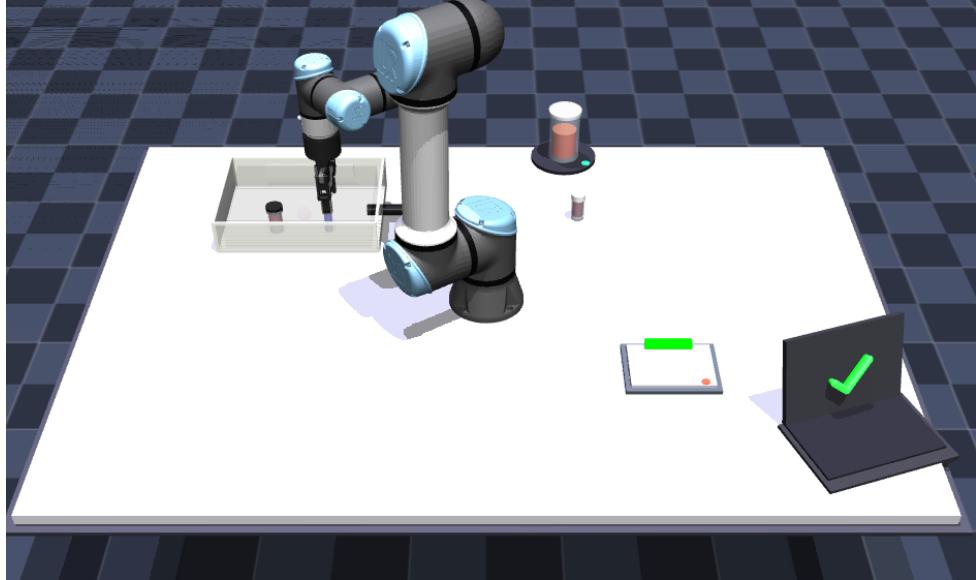


Fig. 4. Mujoco robot simulator executing the commands from Open Roberta.

### 3.3 Treatment Validation

The treatment validation for this study adopts a mixed-methods evaluation approach to assess the effectiveness of the proposed features for guiding users in creating custom reusable components (blocks) within the OpenRoberta environment.

Manuscript submitted to ACM

417     3.3.1 *Participant Recruitment.* A total of 10 participants will be selected to ensure a diverse range of experience levels  
418     with block-based programming. Time constraints and resource availability have influenced the decision to limit the  
419     number of participants. Participants will be recruited from a diverse pool of individuals affiliated with the University  
420     of Southern Denmark and the broader chemistry community. This group of participants includes chemistry teachers,  
421     professional chemical engineers, and students currently enrolled in chemistry-intensive curricula. To ensure relevant  
422     practical expertise, the selection specifically targets those who frequently engage in laboratory environments. The  
423     experimental sessions will be conducted across a range of environments to accommodate participant availability.  
424     Physical sessions will take place within the chemistry laboratories at the University of Southern Denmark (SDU) as  
425     well as a private residential setting. For remote participants, sessions will be administered virtually using Discord for  
426     communication and AnyDesk for remote desktop control.  
427

428     430     *Ethical Considerations and Sampling.* Prior to the commencement of the study, all participants are required to sign a  
429     consent form acknowledging their voluntary participation and granting permission for screen recording and data usage.  
430     It should be noted that this recruitment strategy constitutes *convenience sampling*. As such, they may not represent the  
431     general population.  
432

433     436     3.3.2 *Task Execution.* The participants will initially be given a short introduction to the OpenRoberta UI, as well  
434     437     as the mujoco robot simulator. They will then perform one task which is described by a set of pre-defined steps to  
435     438     perform. This task has been specifically designed to promote the reusability aspect. The task is focused on the domain  
436     439     of chemistry, as it is modelled after a real lab experiment performed by chemistry students at SDU.  
437

438     The participants will be instructed to program the robot to execute the following sequence of operations:  
439

- 440         443         (1) Move the robot arm above mix cylinder
- 441         444         (2) Mix the chemistry ingredients
- 442         445         (3) Move the robot arm above the analysis pad
- 443         446         (4) Analyze the sample
- 444         447         (5) If the solution is analyzed (use if statement) then show a response message in the laptop's screen
- 445         448         (6) Place the following three objects into their corresponding slots in the chemistry equipment toolbox:

- 446             451         • Methanol cylinder
- 447             452         • Chloroform syringe
- 448             453         • Toluene syringe

- 449         454         (7) Important notes for the participants:

- 455             456         • After placing an object to its slot in the toolbox **wait 2 seconds** before you move to pick a new one.
- 456             457         • After placing the **chloroform syringe** to its slot, **move the robot arm up by 10 cm** before you move to pick  
457             the next chemistry object
- 458             459         • Click the **play** button on the bottom right corner to start the simulation
- 460             461         • Click the **reset** button on the bottom right corner to reset the scene of the robot simulator

462     Most optimal solution pre-defined by the researchers:

463     Instead of creating a long linear sequence of blocks, the most optimal solution utilizes a Custom Reusable Component  
464     to handle the repetitive action of placing an object to its corresponding slot inside the equipment toolbox. This approach  
465     not only reduces redundancy but also enhances code maintainability and readability, aligning with best practices in  
466     software development.  
467

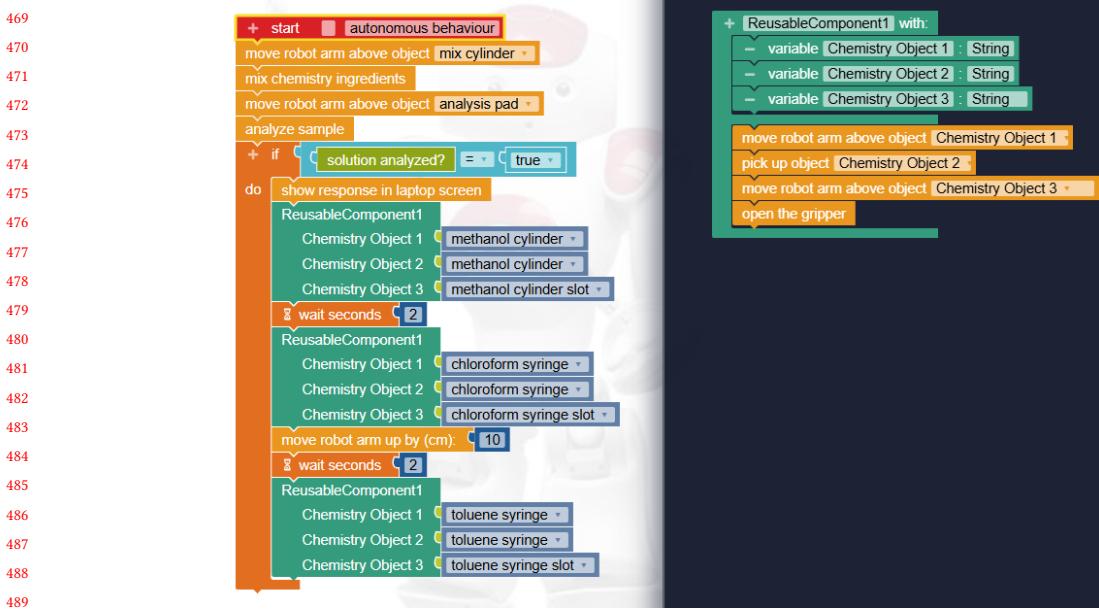


Fig. 5. The optimal solution implemented in OpenRoberta, utilizing a custom block for the object placement sequence.

All the participants will try to complete the task using both the standard and the enhanced version of OpenRoberta. Half of the participants will begin using the enhanced version of OpenRoberta, while the other half will start with the standard version. Participants' interactions with the platform will be observed throughout the task. Guidance will be provided from the researchers to the participants throughout the task.

**3.3.3 Data Gathering and Analysis.** Data collection focuses on both quantitative performance and qualitative feedback from participants:

- (1) **Task Completion Time:** Comparing the participants who will first use the enhanced version of OpenRoberta against those who will first use the standard version.
- (2) **Solution Accuracy:** Evaluated by comparing the participant's block configuration against the pre-defined optimal solution.
- (3) **Survey Feedback:** Collected via a post-experiment survey designed to capture demographic data and subjective perceptions of the effectiveness and usability of the block creation guidance feature.

This comprehensive evaluation will provide a detailed understanding of how useful and effective is the block creation guidance feature to the end-users.

## 4 Results

The treatment validation was concluded with a total of 10 participants. The analysis of the collected data combines quantitative metrics regarding user preference and satisfaction with qualitative feedback derived from survey responses.

Manuscript submitted to ACM

521    **4.1 Performance Evaluation**

522  
523    To evaluate the efficiency and effectiveness of the proposed reusable component features, we analyzed two primary  
524    metrics: Task Completion Time and Solution Accuracy.

525  
526    *4.1.1 Task Completion Time.* The total time required to complete the experimental task was recorded for both the  
527    participants that first used the *Standard* version of OpenRoberta and for those who used the *Enhanced* version first.

528    We compared the performance of participants based on the order of conditions (see Table 2). The analysis reveals a  
529    significant reduction in task duration when using the Enhanced version. The average completion time for the participants  
530    that used the Enhanced version first was 8.5 minutes, compared to 10 minutes for the Standard version.

$$532 \quad \text{Efficiency Improvement} = \frac{10.0 - 8.5}{10.0} \times 100\% = 15\% \quad (1)$$

533  
534

535  
536    Table 2. Breakdown of Mean Task Completion Times

537

538 <b>Experimental Condition</b>	539 <b>Mean Time (min)</b>	540 <b>Efficiency</b>
540 <i>Participants who used Enhanced Version First</i>	541    8.5	541    15%
541 <i>Participants who used Standard Version First</i>	542    10.0	542    —

543  
544    *4.1.2 Solution Accuracy.* Solution accuracy was evaluated by comparing participant solutions against the optimal  
545    reference solution defined in the treatment validation (see Section 3.3).

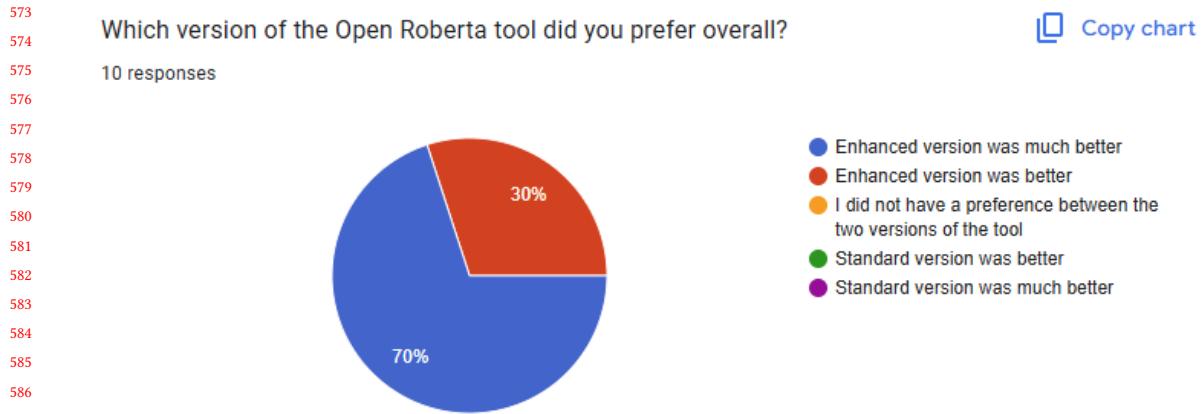
546  
547    *Adoption of Reusable Blocks.* A key metric was the voluntary adoption of the custom reusable component. In the  
548    Enhanced version, 10/10 participants successfully implemented a custom reusable block to handle the repetitive object  
549    placement steps. In contrast, in the Standard condition, participants predominantly relied on linear, repetitive code  
550    structures. Without the guidance features, none of them recognized the opportunity to create a reusable block.

553  
554    **4.2 Survey Quantitative Results**

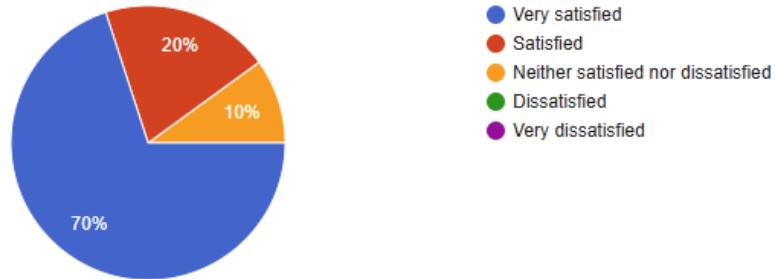
555  
556    *4.2.1 User preference between Standard and Enhanced Versions of OpenRoberta.* The survey results indicate a unanimous  
557    preference for the enhanced version of the OpenRoberta Lab. As illustrated in Figure 6, 70% of participants rated the  
558    enhanced version as “much better” than the standard version, while the remaining 30% rated it as “better.” No participants  
559    preferred the standard version or rated the two versions as equivalent.

560  
561    *4.2.2 Usability of the Guidance Feature.* Regarding usability of the enhanced OpenRoberta version, we received high  
562    acceptance scores. As illustrated in Figure 7, 40% of participants found the enhanced version “very easy” to use, and  
563    60% rated it as “easy.” No participants rated the enhanced version as “Neither easy nor difficult,” “Difficult,” or “Very  
564    difficult” to use.

565  
566    *4.2.3 Evaluation of the Visual Highlighting.* A key component of the enhanced version was the visual highlighting  
567    designed to guide the user into an automatic custom reusable block creation. As shown in Figure 8, results showed a  
568    high level of user satisfaction, with 90% of participants reporting they were either “satisfied” (20%) or “very satisfied”  
569    (70%) with the features. Only one participant (10%) expressed a neutral stance.



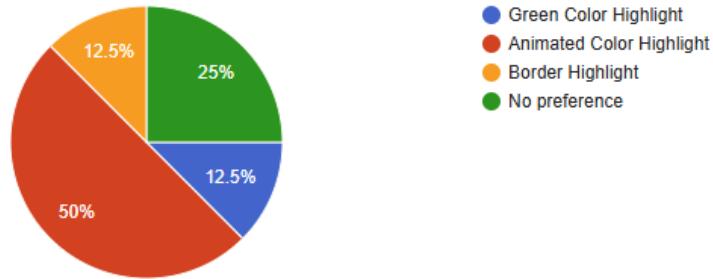
625 How satisfied were you with the visual highlight?  
 626  
 627 10 responses  
 628  
 629  
 630  
 631  
 632  
 633  
 634  
 635  
 636  
 637  
 638  
 639  
 640  
 641



[Copy chart](#)

Fig. 8. Summary of participant responses regarding evaluation of the visual highlighting in the enhanced version of OpenRoberta

642 Which highlight option did you prefer the most?  
 643  
 644  
 645 8 responses  
 646  
 647  
 648  
 649  
 650  
 651  
 652  
 653  
 654  
 655  
 656  
 657  
 658  
 659  
 660  
 661  
 662



[Copy chart](#)

Fig. 9. Summary of participant responses regarding visual highlighting style preference in the enhanced version of OpenRoberta

666 *Efficiency and Speed.* When asked to identify the biggest difference between the two versions, the majority of  
 667 participants cited *efficiency*. Responses frequently described the enhanced version as “faster”. This aligns with the  
 668 quantitative preference data, suggesting that the reusability features successfully reduced the perceived workload.  
 669

670 *Suggestions for Improvement.* Participants also provided constructive feedback regarding the function blocks. Two  
 671 participants specifically suggested that the system should more clearly “*specify parameter names*” within the function  
 672 blocks to improve clarity. Another participant noted that the function call block should be pre-configured for immediate  
 673 use in the blockchain. These suggestions highlight a need for clearer labeling in future iterations of the interface.  
 674

**677 5 Discussion**

**678 5.1 Implications for Theory**

**680 5.1.1 Attention Investment Model. lol**

**681**

**682 5.2 Lessons Learned**

**683 Utilizing OpenRoberta Lab as a representative block-based robotics environment, this study examined the efficacy of**  
**684 automated guidance mechanisms in promoting software reuse among chemistry students and educators engaged in**  
**685 laboratory experimentation.**

**686**

**687 5.2.1 Overall User Preference.** Based on the feedback from the participants, as well as observations of how they solved  
**688 the task, the participants found the enhanced version of OpenRoberta Lab to be better than the standard version.**  
**689 Noteably, 9 out of 10 participants commented on how the enhanced version let them perform their task faster. As**  
**690 described in section 2, this is also one of the main benefits of reuse in the field of software engineering.**

**691**

**692 5.2.2 Overcoming the Recognition Barrier for Reuse.** A defining finding of this study is the contrast in adoption rates:  
**693 100% of participants utilized reusable blocks in the Enhanced version of OpenRoberta Lab, compared to 0% in the**  
**694 Standard.** This confirms the literature cited in Section 1 regarding the high barrier to entry for "Citizen Developers".  
**695 Despite the task being repetitive by design, participants in the standard environment prioritized immediate task**  
**696 completion over code optimization (linear programming).** The Enhanced version successfully shifted this behavior not  
**697 by forcing reuse, but by lowering the cognitive cost of identifying opportunities.** This suggests that for domain experts  
**698 like chemists, the barrier to reuse is not a lack of utility, but a lack of recognition.**

**699**

**700 5.2.3 Impact of Automated Construction of Reusable Components.** The 15% reduction in task completion time highlights  
**701 the value of automating the custom block creation process.** In the standard environment, creating a reusable component  
**702 requires a manual, multi-step process of defining a custom block.** But without creating a custom reusable component, the  
**703 end-users had to manually import the repetitive block sequences to the workspace.** The enhanced version streamlined  
**704 this by automating the structural setup of the custom block once a duplicate sequence of blocks was detected.**

**705**

**706 5.2.4 Importance of Visual Highlighting.** The user preference for the *Animated Color Highlight* (50% preference) and  
**707 the high satisfaction rates (90% satisfied/very satisfied) for the visual highlight underscore the importance of clear**  
**708 visual signals to the end users.** In a dense visual environment like OpenRoberta, static cues are easily overlooked. The  
**709 dynamic nature of the animation highlight acted as a "Just-in-Time" trigger, interrupting the user's tunnel vision exactly**  
**710 when the redundancy occurred.** This supports the use of proactive, visually distinct interruptions in educational block  
**711 based environments to successfully guide and take the attention of the end users.**

**712**

**713 5.2.5 Suggestions by Participants.** Changes suggested by the participants mainly focus on smaller customizations of  
**714 the tool and the OpenRoberta Lab UI.** It would be amiss to claim that the lack of suggested changes, focused on the tool  
**715 overall, indicate that there is no need for improvement of the tool.** As many of the participants consider themselves  
**716 'beginners' in regards to Computer Programming,** it's likely that they lack ideas about other ways the tool could have  
**717 been designed. Instead, these answers can be interpreted as the participants having little to no issue with the current**  
**718 design.**

**719**

### 729 5.3 Implications for Practice

730 The findings of this study have broader implications for the design of End-User Development (EUD) environments and  
731 educational technology. The success of the enhanced OpenRoberta interface suggests three key shifts for future tool  
732 development:

733 *5.3.1 Transitioning from Passive to Proactive Environments.* Current block-based environments (such as Scratch or  
734 standard OpenRoberta) largely rely on a *passive* interaction model, where advanced features like "Functions" sit in a  
735 toolbox waiting to be discovered. Our study demonstrates that domain experts (e.g., chemists) often fail to utilize these  
736 features voluntarily, even when they would be beneficial. The 100% adoption rate in the Enhanced condition implies that  
737 EUD tools must evolve into *active assistants*. Development environments should incorporate background monitoring  
738 systems that detect inefficient patterns (such as code duplication) and proactively intervene with suggestions.

739 *5.3.2 Learning by Example.* Beyond just making the task faster, the tool also acted as a teaching aid. By pointing out  
740 the repetitive block sequences and showing how to improve that, the tool created a "learning moment" exactly when  
741 the user needed it. This suggests that automation reuse assistants can have two benefits: they help domain experts  
742 work faster, but they also teach beginners difficult concepts like how to organize blocks of code by simply showing it to  
743 them in practice.

### 744 5.4 Threats to Validity

745 *5.4.1 External Validity - Convenience Sampling.* The participants to the study were either acquaintances of one of the  
746 authors of the study, or were recruited through these acquaintances. As the study is focused on the domain of chemistry  
747 rather than general use, this did lead to a smaller sample pool of possible participants. As such, the results of this study  
748 cannot be generalized.

749 *5.4.2 Internal Validity - Learning Effect.* As all participants complete the same task twice, it is inevitable that there  
750 will be a carryover effect in regards to learning. This can affect some of the results, such as task completion time. Two  
751 things were done to mitigate this. Firstly, experiment sequence was mixed between subjects. If the first participant first  
752 completed the task using the standard OpenRoberta, then the enhanced version, then the second participant would do  
753 it in the opposite order. Secondly, the experiments were balanced, meaning half of the participants did the standard  
754 -> enhanced sequence, while the other half did enhanced -> standard. In this way, the overall effect on the results is  
755 minimized.

756 *5.4.3 Construct Validity - Ambiguous Metrics.* Certain questions in the survey were too ambiguous to provide valid  
757 measurements. This included asking participants how 'easy' they found the enhanced version of OpenRoberta, what  
758 level of programming knowledge they possess and whether they were satisfied with the highlights. To mitigate this,  
759 another questionnaire was created, based on [XX], which the participants were asked to answer less than one week  
760 post-experiment.

761 *5.4.4 Interpretive Validity - Ambiguous Answers.* Certain answers to the open-ended questions were ambiguous. The  
762 most prominent example is the question about identified differences in the two versions of OpenRoberta. While  
763 multiple participants talked about 'it' being faster, they failed to explicitly mention which version of OpenRoberta  
764 they were referring to. This was mitigated by asking the participants to clarify which version they were referring to  
765 post-experiment.

781        5.4.5 *Descriptive Validity - Limitations to observation.* Due to constraints with time and flexibility between the authors  
 782 and the participants, only one of the authors was present to observe the participants during the experiment. To ensure  
 783 that data from the observation was not affected by this, a screen recording of each participant performing the task was  
 784 saved. Several of the authors reviewed and discussed these recordings together to extract data.  
 785

786

## 787 References

- 788 [1] Felix Adler, Gordon Fraser, Eva Gründinger, Nina Körber, Simon Labrenz, Jonas Lerchenberger, Stephan Lukasczyk, and Sebastian Schweikl. 2021.  
 789        Improving Readability of Scratch Programs with Search-Based Refactoring. In *Proceedings of the IEEE/ACM 43rd International Conference on Software*  
 790        *Engineering: Companion Proceedings (ICSE-SEET)*. IEEE. [doi:10.1109/ICSE-Companion.2021.00105](https://doi.org/10.1109/ICSE-Companion.2021.00105)
- 791 [2] Len Bass, Paul Clements, and Rick Kazman. 2021. *Software Architecture in Practice, 4th Edition*. Addison-Wesley Professional.
- 792 [3] Alexander Bock and Ulrich Frank. 2021. Low-Code Platform. *Business and Information Systems Engineering* 63 (2021). [doi:10.1007/s12599-021-00726-8](https://doi.org/10.1007/s12599-021-00726-8)
- 793 [4] Yuhan Lin and David Weintrop. 2021. The Landscape of Block-Based Programming: Characteristics of Block-Based Environments and How They  
 794        Support the Transition to Text-Based Programming. *Journal of Computer Languages* 67 (2021), 101075. [doi:10.1016/j.jola.2021.101075](https://doi.org/10.1016/j.jola.2021.101075)
- 795 [5] Hugo Lourenço, Carla Ferreira, and João Costa Seco. 2021. OSTRICH - A Type-Safe Template Language for Low-Code Development. In *2021 ACM/IEEE*  
 796        *24th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. 216–226. [doi:10.1109/MODELS50736.2021.00030](https://doi.org/10.1109/MODELS50736.2021.00030)
- 797 [6] Vlad Magdalin. 2012. Low code platform tool Webflow. <https://webflow.com/>.
- 798 [7] Derek Roos. 2005. Low code platform tool Mendix. <https://www.mendix.com/>.

799 Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009

800

801

802

803

804

805

806

807

808

809

810

811

812

813

814

815

816

817

818

819

820

821

822

823

824

825

826

827

828

829

830

831

832