Title 3

ANNE-MARIE ROMMERDAHL, SDU, Denmark
JEREMY ALEXANDER RAMÍREZ GALEOTTI, SDU, Denmark
DIMITRIOS DAFNIS, SDU, Denmark
NASIFA AKTER, SDU, Denmark
MOHAMMAD HOSEIN KARDOUNI, SDU, Denmark

A clear and well-documented LaTeX document is presented as an article formatted for publication by ACM in a conference proceedings or journal publication. Based on the “acmart” document class, this article presents and explains many of the common variations, as well as many of the formatting elements an author may use in the preparation of the documentation of their work.

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.

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

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

1 Introduction

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 it's '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 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 occured, with domain-specific reference artifacts being categorized as 'rare'. Most studied systems offered catalogs of "reusable functions or examples of predefined

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.

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.

There have been proposed some ideas on how to promote reuse for LCPs, such as the strongly-typed rich templating language OSTRICH, developed for the model-driven low-code platform OutSystems[6]. OutSystems provides scaffolding mechanisms for common development patterns and sample screen templates, both designed by experts on domain-specific languages (DSL). The practice of using templates in the OutSystems platform involves cloning and modifying samples, which may require more knowledge than the end-user posseses. The goal of OSTRICH is to remove this need for adapation when using templates, to remove the knowledge-barrier when making use of the available templates. This is done by abstracting and parameterizing the templates. A limitation of OSTRICH, is that it currently only supports the top nine most used production-ready screen templates from OutSystems. The end-user may not create and save their own templates, nor can they re-apply a template which they have customized.

Another approach focused on enabling reuse of models, by converting and merging models into a single graph (the Knowledge Graph), which acts as a repository of models[4]. This graph is used to provide recommendations to the end-user, based on the model they're currently building. While this feature of recommending models (either constructed by domain experts and then developed by model experts, or made by the end-user themselves) could prove very useful, the study is clearly not focused on guiding the user towards reusing their own models.

Building on the ideas discussed for improving reuse in low-code development platforms (LCDPs), several popular tools show these concepts in action. For instance, Webflow[7] is a leading low-code platform that offers a wealth of features for building responsive websites. One of its standout features is the ability to create reusable components and UI kits, which can significantly speed up the development process. With Webflow's intuitive interface, developers can quickly design and prototype components, and then reuse them across multiple pages and projects. Despite all of the useful features that this tools has, it does not provide guidance to the end-users to create custom reusable components.

In a similar way, Mendix[8] takes this further for full enterprise apps by offering shareable building blocks like simple actions (microflows) and UI parts that anyone on a team can grab and use again without recoding. Through its Marketplace, a free online hub, you can download ready templates, connectors for tools like Salesforce, and basic setups that fit right into new projects, making everything faster and more uniform. This approach builds on the flexibility seen in platforms like Webflow, but adds strong team tools and AI suggestions to spot and create reusable pieces, empowering even beginners to build complex apps while keeping reuse simple and widespread. This tool does offer guidance for the end-users to create custom reusable components through its AI suggestions, a lot of times these suggestions are not accurate enough (how do we know this??**).

OutSystems[9] further enhances the concept of reuse in low-code development platforms by emphasizing rapid application delivery through its robust set of features. Like Webflow and Mendix, OutSystems also provides a library of reusable components and templates that help developers complete projects faster. Its user-friendly visual development environment allows users to easily drag and drop elements while connecting with existing systems. OutSystems also supports teamwork with built-in version control and feedback features, making it easy for teams to share and improve reusable components. Additionally, the platform uses AI to suggest the best solutions and components for specific tasks. By encouraging reuse at both individual and team levels, OutSystems enables organizations to create scalable applications quickly while ensuring quality and consistency. Similarly to the previous tool explained, the AI suggestions that this tool provides are not always accurate to successfully guide the end-user to create custom reusable components (again, how do we know this?**).

In order to analyze how block-based robotics environments address reuse area, 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 and improve user productivity.

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.

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 [5]—such as abstraction or reuse—can enhance learning outcomes. In contrast, our work emphasizes guided abstraction, helping users understand and practice modular design directly within block-based environments.

Techapalokul and Tilevich (2019) proposed extending the Scratch programming environment with facilities for reusing individual custom blocks to promote procedural abstraction and improve code quality. They observed that while Scratch enables remixing of entire projects, it lacks mechanisms for reusing smaller, modular pieces of code. Their work suggests that supporting such fine-grained code reuse could enhance programmer productivity, creativity, and learning outcomes. Building on this idea, our project applies similar principles within the OpenRoberta environment by automating the detection of duplicate code segments and guiding users toward creating reusable custom blocks. 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. Users frequently create repetitive code because they struggle to recognize duplicate patterns, lack knowledge about abstraction mechanisms, or find existing tools too complex to use effectively. This problem manifests in several ways: programs become unnecessarily long and difficult to maintain and small changes require modifications in multiple locations, increasing the risk of errors. 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. 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 necessarry 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.
(2) **Backend (Flask Server):** The client transmits these headers via HTTP POST requests to a Flask-based API Endpoint. A "Translator" component processes the data, mapping the abstract block definitions to concrete Python methods compatible with the robot's control logic.
(3) **Simulation (Mujoco):** The mapped methods trigger the execution of commands within the Mujoco Simulator, which renders the physical behavior of the cobot in the virtual environment.

3.2.3 Detection Algorithm. The core of the reuse assistance, located on the Client Side, is the sequence detection algorithm encapsulated in the `highlightOnlyFunctionCandidates` function. The algorithm operates in several steps:

- **Linearization:** It first converts the hierarchical block structure into a linear chain of significant operational blocks, filtering out simple literals to focus on logic and action blocks.
- **Pattern Key Generation:** For a sliding window of block sequences (ranging from a minimum to a maximum length), it generates a unique "structural pattern key." This key is a hash or string representation of the block types and their connectivity, ignoring specific parameter values.
- **Pattern Matching:** The algorithm aggregates sequences by identical pattern keys. If a pattern key appears more than once (frequency ≥ 2), it is flagged as a candidate for reuse.
- **Parameter Extraction:** Once a duplicate group is identified, the `extractLiteralParameters` function compares the instances to identify varying literals. These variations are mapped to future function parameters, ensuring the created abstraction is generalized correctly.

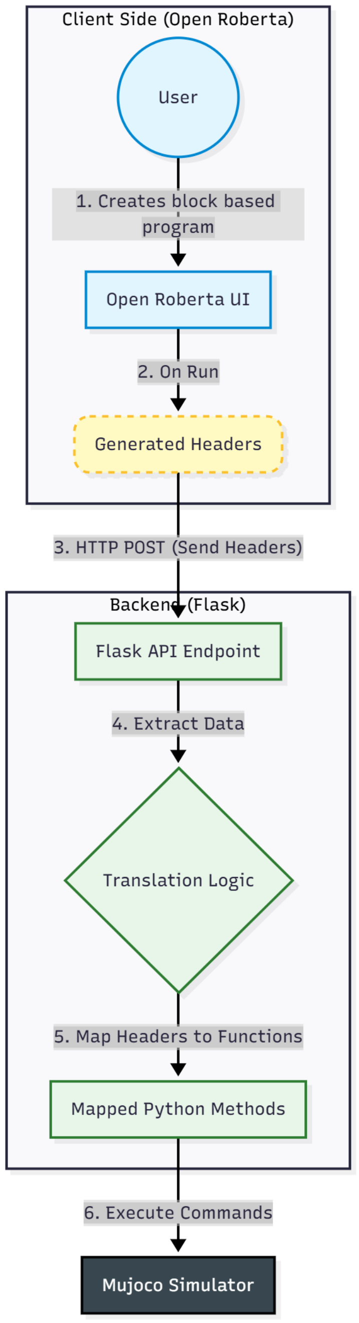


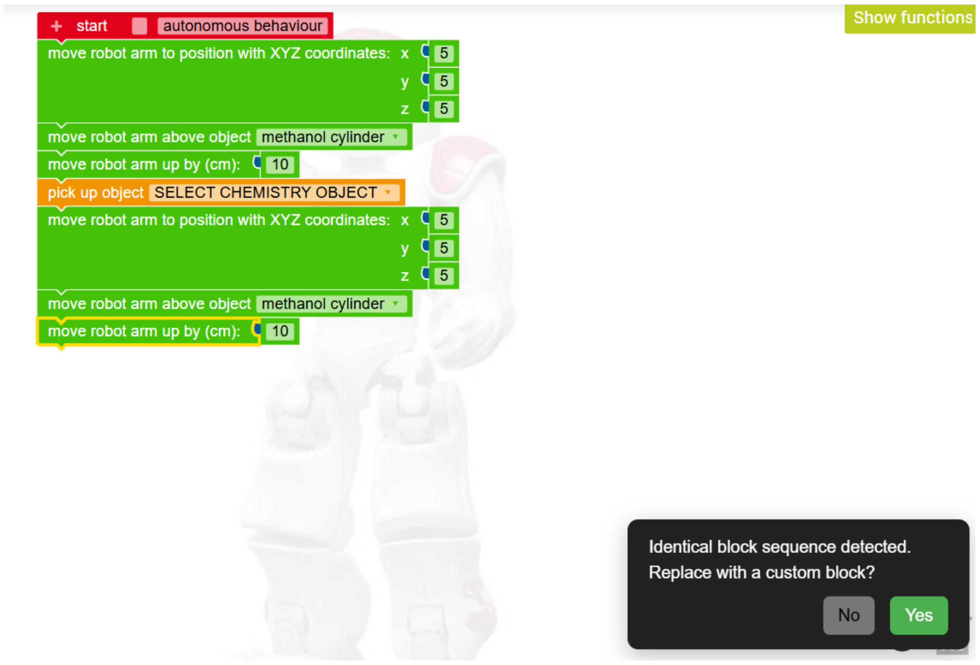
Fig. 1. System architecture

Algorithm 1 Sequence Detection and Pattern Key Generation

```
Require: $Blocks$: List of blocks in workspace
Ensure: $Candidates$: List of duplicate sequences
 1: $LinearChain \leftarrow$ Linearize($Blocks$)
 2: $Patterns \leftarrow$ Map⟨String, List⟨Sequence⟩⟩
 3: for $len \leftarrow$ MinLen to MaxLen do
 4:   for $i \leftarrow 0$ to length($LinearChain$) $- len$ do
 5:     $sequence \leftarrow LinearChain[i : i + len]$
 6:     $sequenceKey \leftarrow$ “”
 7:     for all $block$ in $sequence$ do
 8:       $sequenceKey \leftarrow sequenceKey + block.type$
 9:     end for
10:     $Patterns[sequenceKey]$.add($sequence$)
11:   end for
12: end for
13: $Candidates \leftarrow$ []
14: for all $sequenceKey$ in $Patterns$ do
15:   if size($Patterns[sequenceKey]$) $\geq 2$ then
16:     $Candidates$.add($Patterns[sequenceKey]$)
17:   end if
18: end for
19: return $Candidates$
```

Algorithm 1. illustrates the core logic for identifying reusable candidates by abstracting literal values.

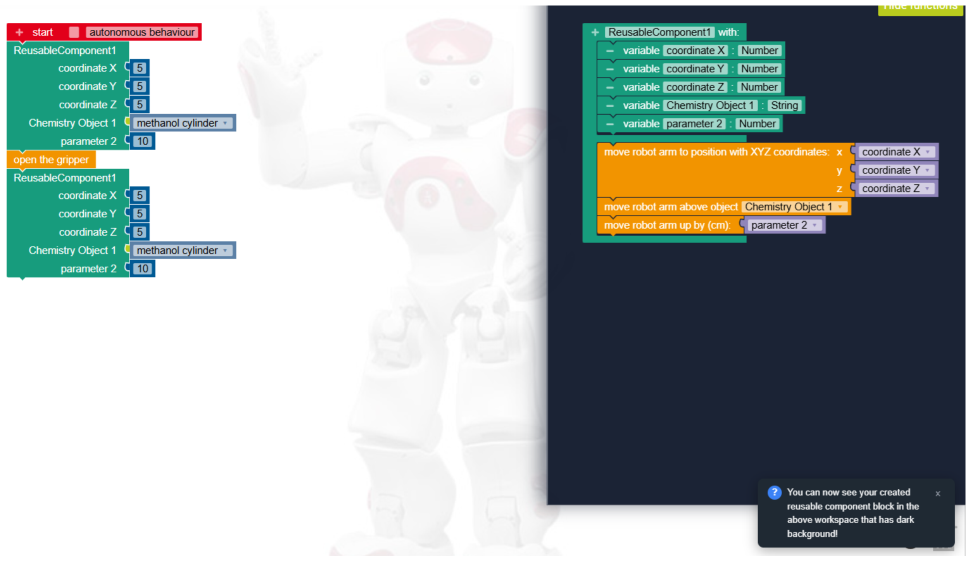
3.2.4 User Interface and Interaction. The user interface is designed to be intuitive and non-disruptive. When the detection algorithm identifies a candidate, the system visually highlights the blocks on the canvas as illustrated in Figure 2. A non-blocking toast notification appears, prompting the user to confirm the refactoring. If confirmed, the system automatically generates the custom block definition in a dedicated workspace area (handling visibility via `revealDefinitionWorkspacePane`) and updates the main workspace, replacing the redundant code with concise function calls as shown in Figure 3. This process abstracts the complexity of manual function creation, guiding the user toward modular design practices.



```
+ start  autonomous behaviour
move robot arm to position with XYZ coordinates: x 5
                                                 y 5
                                                 z 5
move robot arm above object methanol cylinder
move robot arm up by (cm): 10
pick up object SELECT CHEMISTRY OBJECT
move robot arm to position with XYZ coordinates: x 5
                                                 y 5
                                                 z 5
move robot arm above object methanol cylinder
move robot arm up by (cm): 10
```

Identical block sequence detected.
Replace with a custom block?

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



```
+ start  autonomous behaviour
ReusableComponent1
        coordinate X 5
        coordinate Y 5
        coordinate Z 5
   Chemistry Object 1 methanol cylinder
         parameter 2 10
open the gripper
ReusableComponent1
        coordinate X 5
        coordinate Y 5
        coordinate Z 5
   Chemistry Object 1 methanol cylinder
         parameter 2 10
```

```
+ ReusableComponent1 with:
  − variable coordinate X : Number
  − variable coordinate Y : Number
  − variable coordinate Z : Number
  − variable Chemistry Object 1 : String
  − variable parameter 2 : Number
  move robot arm to position with XYZ coordinates: x coordinate X
                                                   y coordinate Y
                                                   z coordinate Z
  move robot arm above object Chemistry Object 1
  move robot arm up by (cm): parameter 2
```

You can now see your created reusable component block in the above workspace that has dark background!

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

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.

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

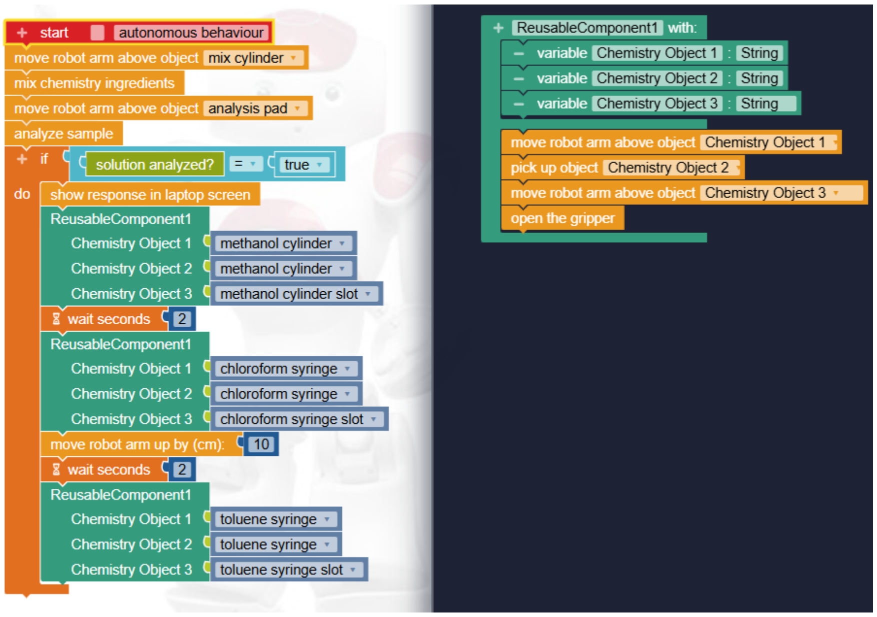
Ethical Considerations and Sampling. Prior to the commencement of the study, all participants are required to sign a consent form acknowledging their voluntary participation and granting permission for screen recording and data usage. It should be noted that this recruitment strategy constitutes *convenience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

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

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

(1) Move the robot arm above mix cylinder
(2) Mix the chemistry ingredients
(3) Move the robot arm above the analysis pad
(4) Analyze the sample
(5) If the solution is analyzed (use if statement) then show a response message in the laptop's screen
(6) Place the following three objects into their corresponding slots in the chemistry equipment toolbox:
    - Methanol cylinder
    - Chloroform syringe
    - Toluene syringe
(7) Important notes for the participants:
    - *After placing an object to its slot in the toolbox* ***wait 2 seconds*** *before you move to pick a new one.*
    - *After placing the* ***chloroform syringe*** *to its slot,* ***move the robot arm up by 10 cm*** *before you move to pick the next chemistry object*
    - *Click the* ***play*** *button on the bottom right corner to start the simulation*
    - *Click the* ***reset*** *button on the bottom right corner to reset the scene of the robot simulator*

Most optimal solution pre-defined by the researchers:



```
start  autonomous behaviour
move robot arm above object  mix cylinder
mix chemistry ingredients
move robot arm above object  analysis pad
analyze sample
if  solution analyzed? = true
do  show response in laptop screen
    ReusableComponent1
        Chemistry Object 1  methanol cylinder
        Chemistry Object 2  methanol cylinder
        Chemistry Object 3  methanol cylinder slot
    wait seconds  2
    ReusableComponent1
        Chemistry Object 1  chloroform syringe
        Chemistry Object 2  chloroform syringe
        Chemistry Object 3  chloroform syringe slot
    move robot arm up by (cm):  10
    wait seconds  2
    ReusableComponent1
        Chemistry Object 1  toluene syringe
        Chemistry Object 2  toluene syringe
        Chemistry Object 3  toluene syringe slot
```

```
ReusableComponent1 with:
    variable  Chemistry Object 1 : String
    variable  Chemistry Object 2 : String
    variable  Chemistry Object 3 : String
    move robot arm above object  Chemistry Object 1
    pick up object  Chemistry Object 2
    move robot arm above object  Chemistry Object 3
    open the gripper
```

Fig. 4. The optimal solution implemented in OpenRoberta, utilizing a custom block for the object placement sequence.

Instead of creating a long linear sequence of blocks (hard-coding the movement for all three objects), the most optimal solution utilizes a **Custom Reusable Component** to handle the repetitive action of placing an object to its corresponding slot inside the equipment toolbox. This approach not only reduces redundancy but also enhances code maintainability and readability, aligning with best practices in software development.

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 utility of the block creation guidance features.

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.

4.1 Performance Evaluation

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

4.1.1 Task Completion Time. The total time required to complete the experimental task was recorded for both the *Standard* and *Enhanced* conditions.

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

$$\text{Efficiency Improvement} = \frac{10.0 - 8.5}{10.0} \times 100\% = 15\% \tag{1}$$

Table 2. Breakdown of Mean Task Completion Times

Experimental Condition	**Mean Time (min)**
Group of Participants that used the Enhanced OpenRoberta Version First	8.5
Group of Participants that used the Standard OpenRoberta Version First	10.0

4.1.2 Solution Accuracy. Solution accuracy was evaluated by comparing participant solutions against the optimal reference solution defined in the treatment evaluation.

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

4.2 Survey Quantitative Results

4.2.1 User preference between Standard and Enhanced Versions of OpenRoberta. The survey results indicate a unanimous preference for the enhanced version of the OpenRoberta Lab. As illustrated in Figure 5, 70% of participants rated the enhanced version as "much better" than the standard version, while the remaining 30% rated it as "better." No participants preferred the standard version or rated the two versions as equivalent.

4.2.2 Usability of the Guidance Feature. Regarding usability of the enhanced OpenRoberta version, we received high acceptance scores. As illustrated in Figure 6, 40% of participants found the enhanced version "very easy" to use, and 60% rated it as "easy." No participants rated the enhanced version as "Neither easy nor difficult," "Difficult," or "Very difficult" to use.

4.2.3 Evaluation of the Visual Highlighting. A key component of the enhanced version was the visual highlighting designed to guide the user into an automatic custom reusable block creation. As shown in Figure 7, results showed a

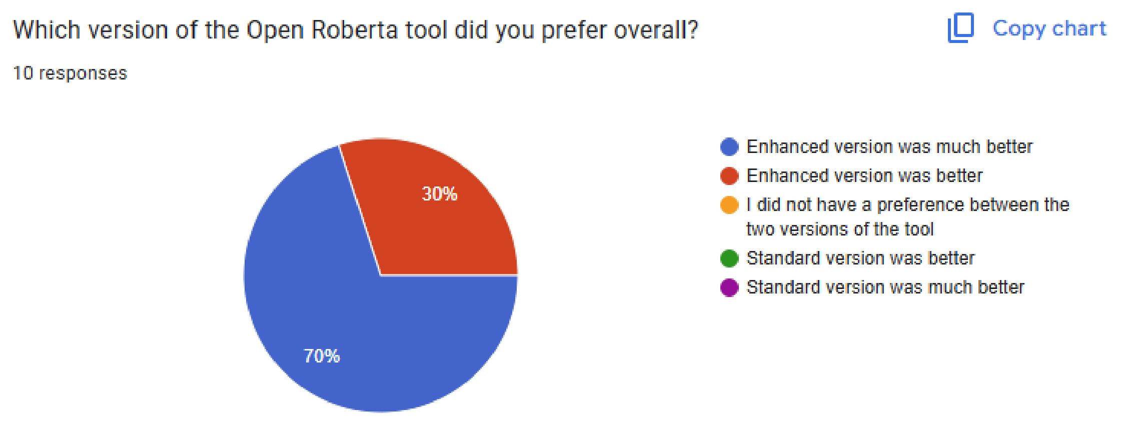


Fig. 5. Summary of participant responses regarding overall preference between the standard and enhanced versions of OpenRoberta

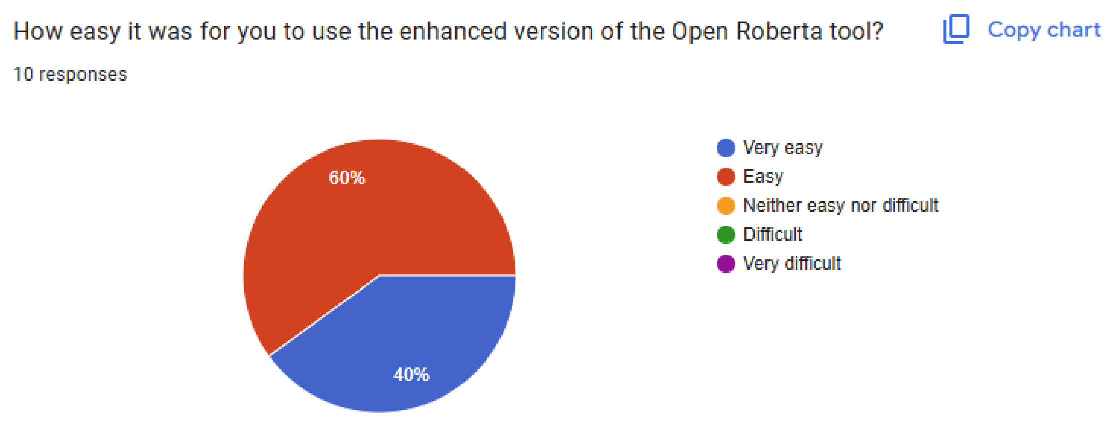


Fig. 6. Summary of participant responses regarding overall preference between the standard and enhanced versions of OpenRoberta

high level of user satisfaction, with 90% of participants reporting they were either "satisfied" (20%) or "very satisfied" (70%) with the features. Only one participant (10%) expressed a neutral stance.

4.2.4 Visual Highlighting Style Preference. When asked about specific highlighting preferences, as depicted in Figure 8 the *Animated Color Highlight* was the most popular choice, preferred by 50% of the users. A significant portion of participants (30%) expressed no strong preference between the styles, suggesting that the presence of guidance was more important than the specific animation style used.

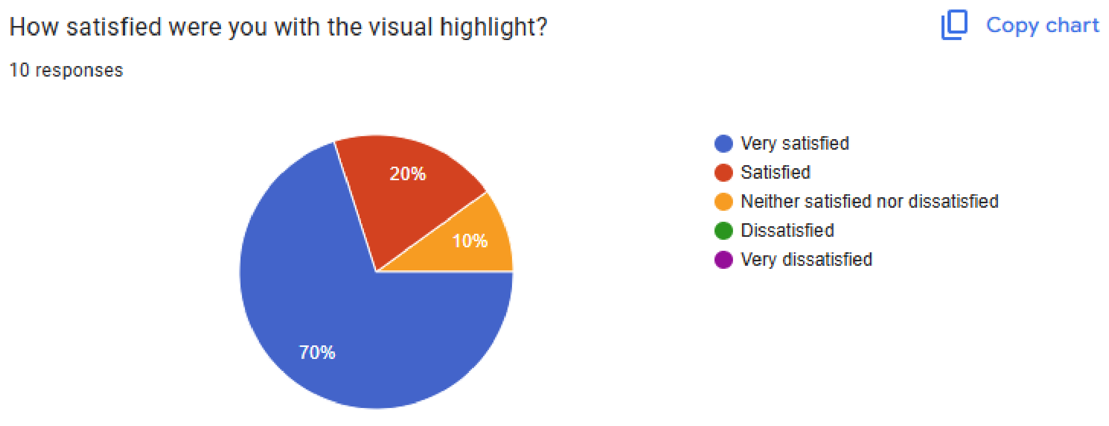


Fig. 7. Summary of participant responses regarding overall preference between the standard and enhanced versions of OpenRoberta

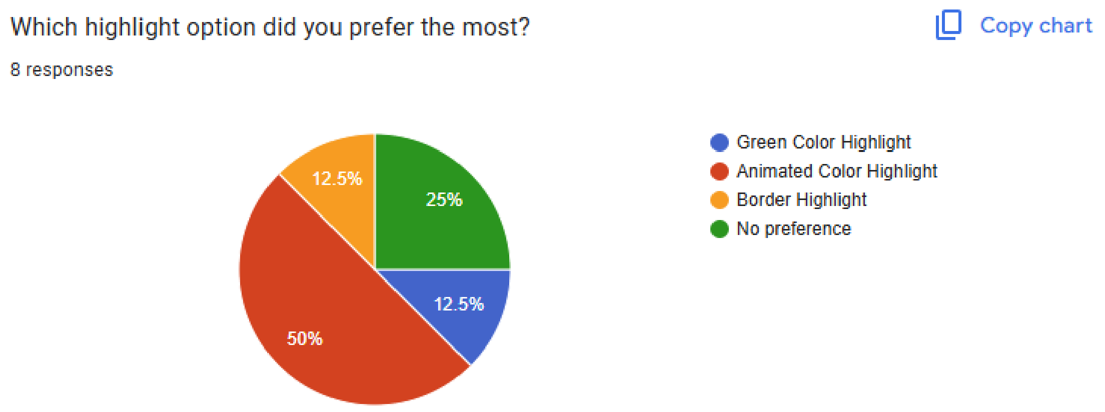


Fig. 8. Summary of participant responses regarding overall preference between the standard and enhanced versions of OpenRoberta

4.3 Qualitative Feedback

The post-experiment survey included open-ended questions to gather detailed feedback. The thematic analysis of these responses revealed two primary findings:

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

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

5 Discussion

5.1 Lessons Learned

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

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

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

5.1.2 Impact of Automated Construction of Reusable Components. The 15% reduction in task completion time highlights the value of automating the block creation process. In the standard environment, creating a reusable component requires a manual, multi-step process of defining a function and relocating blocks. The enhanced version streamlined this by automating the structural setup of the custom block once a duplicate was detected. This confirms that removing the "friction" of manual block assembly is crucial for encouraging reusability among non-programmers.

5.1.3 Visual Salience in Learning. The user preference for the *Animated Color Highlight* (50% preference) and the high satisfaction rates (90% satisfied/very satisfied) underscore the importance of visual salience. In a dense visual environment like OpenRoberta, static cues are easily overlooked. The dynamic nature of the animation acted as a "Just-in-Time" trigger, interrupting the user's tunnel vision exactly when the redundancy occurred. This supports the use of proactive, visually distinct interruptions in educational IDEs to correct inefficient patterns in real-time.

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

5.2 Implications for Practice

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

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

5.2.2 Learning by Example. Beyond just making the task faster, the tool also acted as a teaching aid. By pointing out the repetitive code and showing how to fix it, the tool created a "learning moment" exactly when the user needed it. This suggests that automation tools can have two benefits: they help experts work faster, but they also teach beginners difficult concepts—like how to organize blocks of code and use inputs—simply by showing them a practical example.

5.3 Threats to Validity

5.3.1 Convenience Sampling. The participants to the study were either aquantiances of one of the authors of the study, or were recruited through these aquantiances. As such, the results of this study do not represent the general population within the domain of chemistry.

5.3.2 Limitations to observation. Due to constraints with time and flexibility, only one of the authors was present to observe the participants. To ensure that data from the observation was not affected by this, a screen recording of each participant performing the task was saved. Several of the authors reviewed and discussed these recordings together to extract data.

6 Appendices

If your work needs an appendix, add it before the "`\end{document}`" command at the conclusion of your source document.

Start the appendix with the "`appendix`" command:

```
\appendix
```

and note that in the appendix, sections are lettered, not numbered. This document has two appendices, demonstrating the section and subsection identification method.

7 Multi-language papers

Papers may be written in languages other than English or include titles, subtitles, keywords and abstracts in different languages (as a rule, a paper in a language other than English should include an English title and an English abstract). Use `language=...` for every language used in the paper. The last language indicated is the main language of the paper.

For example, a French paper with additional titles and abstracts in English and German may start with the following command

```
\documentclass[sigconf, language=english, language=german,
               language=french]{acmart}
```

The title, subtitle, keywords and abstract will be typeset in the main language of the paper. The commands `\translatedXXX`, `XXX` begin title, subtitle and keywords, can be used to set these elements in the other languages. The environment `translatedabstract` is used to set the translation of the abstract. These commands and environment have a mandatory first argument: the language of the second argument. See `sample-sigconf-i13n.tex` file for examples of their usage.

8 SIGCHI Extended Abstracts

The "`sigchi-a`" template style (available only in LaTeX and not in Word) produces a landscape-orientation formatted article, with a wide left margin. Three environments are available for use with the "`sigchi-a`" template style, and produce formatted output in the margin:

`sidebar`: Place formatted text in the margin.
`marginfigure`: Place a figure in the margin.
`margintable`: Place a table in the margin.

Acknowledgments

To Robert, for the bagels and explaining CMYK and color spaces.

References

[1] Felix Adler, Gordon Fraser, Eva Gründinger, Nina Körber, Simon Labrenz, Jonas Lerchenberger, Stephan Lukasczyk, and Sebastian Schweikl. 2021. Improving Readability of Scratch Programs with Search-Based Refactoring. In *Proceedings of the IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-SEET)*. IEEE. doi:10.1109/ICSE-Companion.2021.00105

[2] Len Bass, Paul Clements, and Rick Kazman. 2021. *Software Architecture in Practice, 4th Edition.* Addison-Wesley Professional.

[3] Alexander Bock and Ulrich Frank. 2021. Low-Code Platform. *Business and Information Systems Engineering* 63 (2021). doi:10.1007/s12599-021-00726-8

[4] Ilirian Ibrahimi and Dimitris Moudilos. 2022. Towards model reuse in low-code development platforms based on knowledge graphs. In *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings* (Montreal, Quebec, Canada) *(MODELS '22)*. Association for Computing Machinery, New York, NY, USA, 826–836. doi:10.1145/3550356.3561570

[5] Yuhan Lin and David Weintrop. 2021. The Landscape of Block-Based Programming: Characteristics of Block-Based Environments and How They Support the Transition to Text-Based Programming. *Journal of Computer Languages* 67 (2021), 101075. doi:10.1016/j.cola.2021.101075

[6] Hugo Lourenço, Carla Ferreira, and João Costa Seco. 2021. OSTRICH - A Type-Safe Template Language for Low-Code Development. In *2021 ACM/IEEE 24th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. 216–226. doi:10.1109/MODELS50736.2021.00030

[7] Vlad Magdalin. 2012. Low code platform tool Webflow. https://webflow.com/.

[8] Derek Roos. 2005. Low code platform tool Mendix. https://www.mendix.com/.

[9] Paulo Rosado. 2011. Low code platform tool Outsystems. https://www.outsystems.com/.

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