Guiding End-Users Toward Software Reuse: An Evaluation of Automated Assistance in Block-Based Programming for Chemistry Laboratory Automation

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

***Abstract*—Background: End-users who program collaborative robots for laboratory automation often create repetitive code because they struggle to recognize opportunities for reuse. While block-based programming environments provide accessible interfaces, they do not actively guide users toward creating reusable components.**
Objective: This study investigates whether automated guidance can help end-users recognize and apply code reuse practices. We developed the Reuse Assistant, a feature that automatically detects duplicate code sequences in OpenRoberta and guides users to create reusable custom blocks through visual highlighting and one-click refactoring.
Methods: Through a within subjects study with 10 participants from the chemistry domain, we evaluated the feature's impact on performance, usability, and perceived workload.
Results: Automated guidance increased reuse adoption from 0% in the standard environment to 100% with the Reuse Assistant. The feature achieved high usability scores (SUS mean: 84.1) and imposed minimal cognitive burden (NASA-TLX mean: 1.88). A significant order effect revealed that prior manual experience helps users appreciate automation benefits.
Conclusions: Our findings extend the Attention Investment Model and Learning Barriers Framework, demonstrating that proactive assistance can transform feature adoption from a high-cost investment to a low-cost opportunistic choice. The results provide design principles for developing end-user programming tools that effectively guide domain experts without adding complexity.

1 Introduction

Software reuse is a fundamental practice in software engineering, enabling developers to build on existing solutions rather than writing code from scratch. However, end-users who program collaborative robots (cobots) for laboratory automation often lack the knowledge to recognize and apply reuse opportunities. This problem is particularly acute in domains like chemistry, where scientists need to automate repetitive experimental procedures but have limited programming expertise.

Block-based programming environments such as OpenRoberta Lab provide accessible interfaces for programming robots, but they do not actively guide users toward creating reusable components. As a result, end-users frequently produce long, repetitive programs that are difficult to maintain and modify. When experimental protocols change, users must manually update code in multiple locations, increasing the risk of errors and discouraging adoption of automation features.

This study addresses the question: Can automated guidance help end-users recognize and apply code reuse in block-based programming? We developed the Reuse Assistant, a feature that automatically detects duplicate code sequences and guides users to create reusable custom blocks through visual highlighting and one-click refactoring.

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.

Through a within subjects study with 10 participants from the chemistry domain, we evaluated whether proactive automated assistance can overcome the barriers that prevent end-users from adopting reuse practices.

Our investigation examined three research questions: (1) Can the Reuse Assistant improve end-user performance? (2) Is the Reuse Assistant sufficiently usable for end-users? (3) What is the perceived workload when using the Reuse Assistant? The results showed that automated guidance increased reuse adoption from 0% to 100%, achieved high usability scores (SUS mean: 84.1), and imposed minimal cognitive burden (NASA-TLX mean: 1.88).

The contributions of this work are both theoretical and practical. We extend the Attention Investment Model by demonstrating that proactive assistance can transform feature adoption from a high-cost investment to a low-cost opportunistic choice. We also extend Ko et al.'s Learning Barriers Framework by showing that the selection barrier dominates in block-based environments for end-users. From a practical perspective, our results demonstrate that simple design principles (visual highlighting, one-click acceptance, immediate feedback) can effectively guide end-users without adding complexity, as evidenced by high usability and low workload scores.

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 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 processes", but they were found to be generic, or have a limited scope[5]. 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 [7], 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[8]. 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[9], 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[10] 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. 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 approach is intentionally simple so it is easy to read and to implement in a real block editor. The algorithm follows three main steps:

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

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

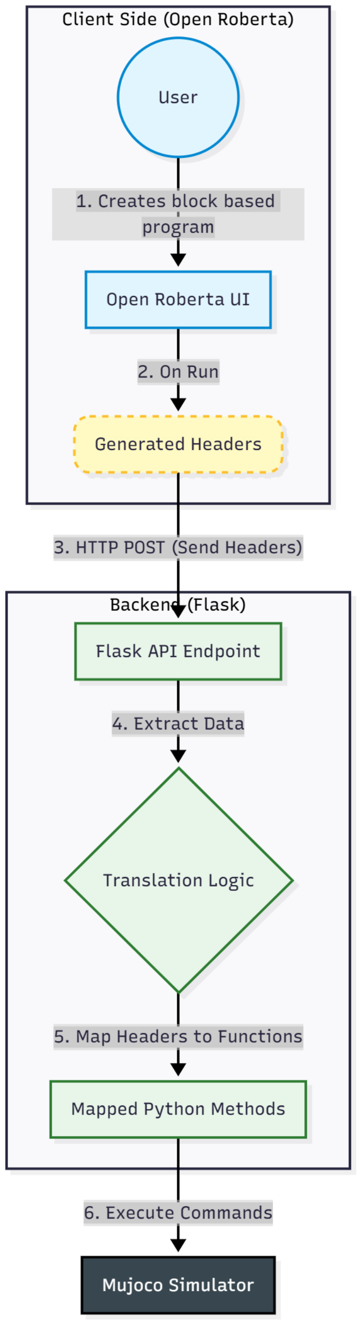


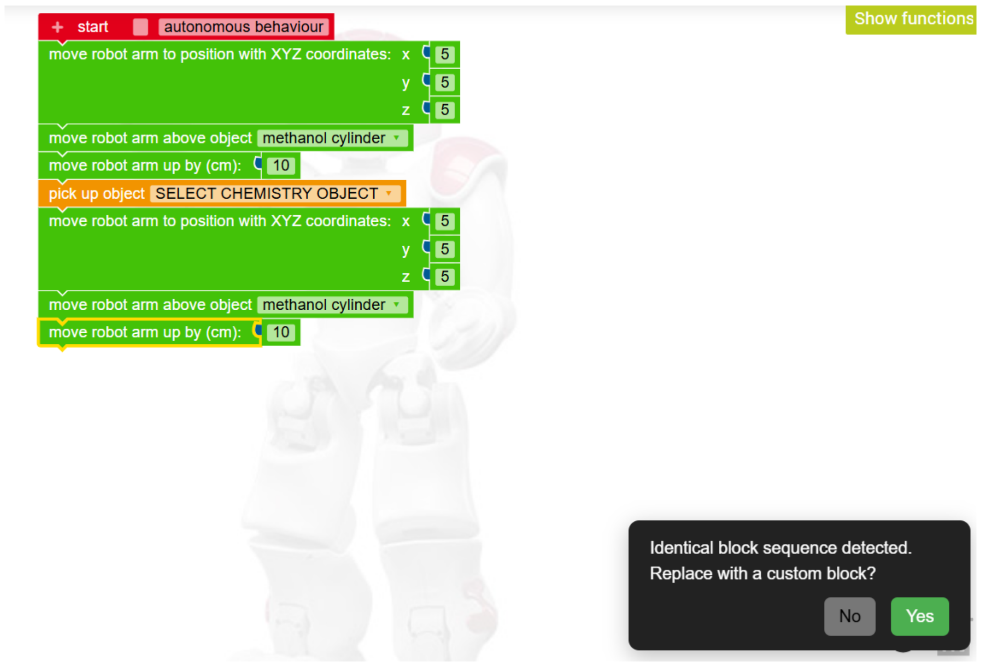
Fig. 1. System architecture

Algorithm 1 Duplicate Sequence Detection

```
Require: Workspace, StartBlock    // user's block workspace
Require: MinimumSequenceLength = 3, MinimumDifferentBlockTypesInSequence = 3, MaxSequenceLength = 10
Ensure: ReusableComponentCandidates    // list of repeated block sequences to return
 1: Chain = buildLinearChain(StartBlock)
 2: Sequences = List⟨sequence⟩
 3: for startIndex = 0 to length(Chain) - 1 do
 4:     for $sequenceLength$ = 1 to $MaxSequenceLength$ do
 5:         sequence = Chain[startIndex .. startIndex + sequenceLength - 1]
 6:         numberOfBlockTypesInSequence = getNumberOfDistinctBlockTypes(sequence)
 7:         if sequenceLength >= MinimumSequenceLength and numberOfBlockTypesInSequence >= MinimumDifferentBlockTypesInSequence then
 8:             Sequences.append(sequence) // record sequence occurrence
 9:         end if
10:     end for
11: end for
12: ReusableComponentCandidates = {Sequences | $occurrence \geq 2$}
13: sort ReusableComponentCandidates by (longest sequence length and most recent occurrence)
14: return ReusableComponentCandidates[0] // Return highest priority candidate
```

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

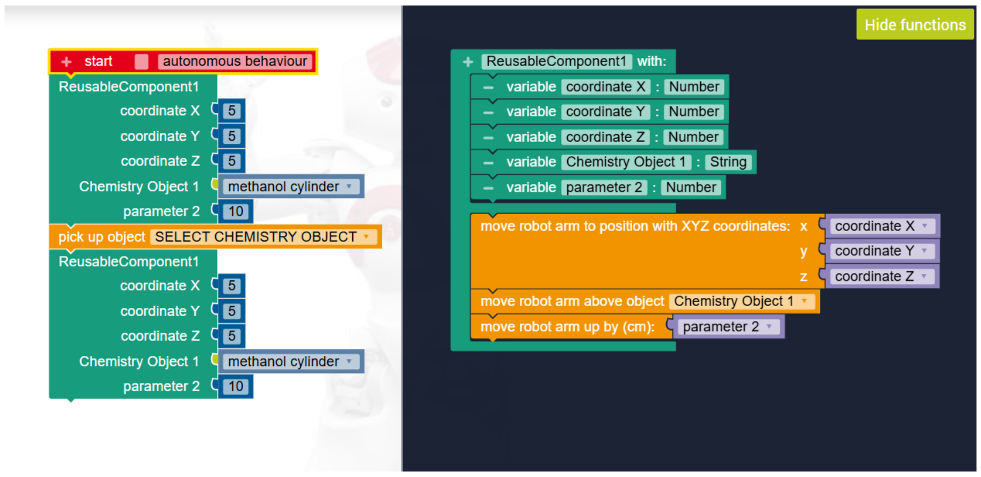
3.2.4 User Interface and Interaction. The user interface is designed to be intuitive and non-disruptive. When the detection algorithm identifies a candidate, the system visually highlights the blocks on the canvas as illustrated in Figure 2. A non-blocking toast notification appears, prompting the user 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. After the user presses the run simulation button, the robot simulator of mujoco opens up and executes the commands provided by the user inside the Open Roberta workspace. This is illustrated in Figure 4.



```
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
pick up object SELECT CHEMISTRY OBJECT
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
```

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

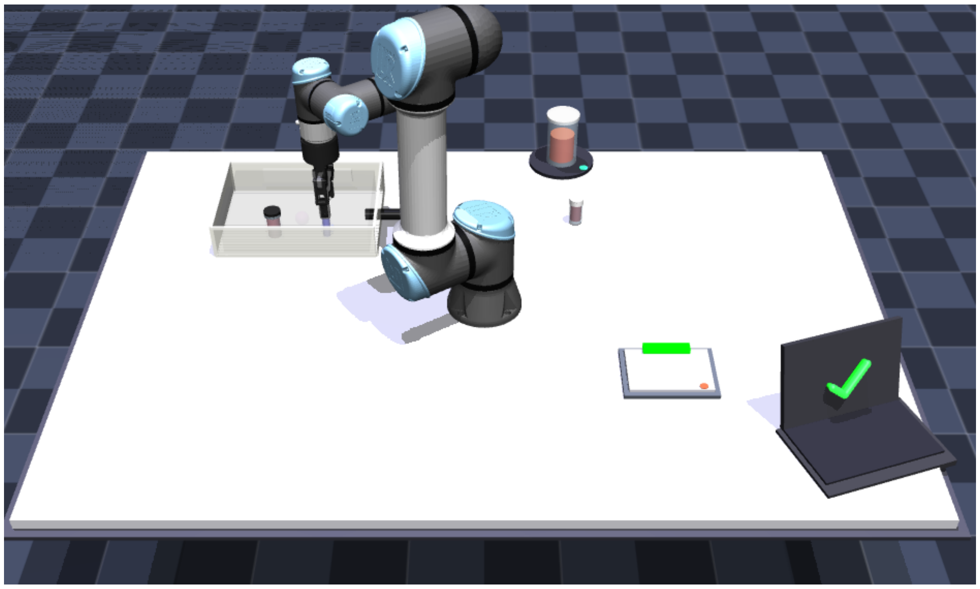


Fig. 4. Mujoco robot simulator executing the commands from Open Roberta.

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.

3.3.1 Participant Recruitment. A total of 10 participants were selected with similar level of expertise in block-based programming. Time constraints and resource availability have influenced the decision to limit the number of participants. Participants were 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 were conducted across a range of environments to accommodate participant availability. Physical sessions took place within the chemistry laboratories at the University of Southern Denmark (SDU) as well as a private residential setting. For remote participants, sessions were administered virtually using Discord for communication and AnyDesk for remote desktop control.

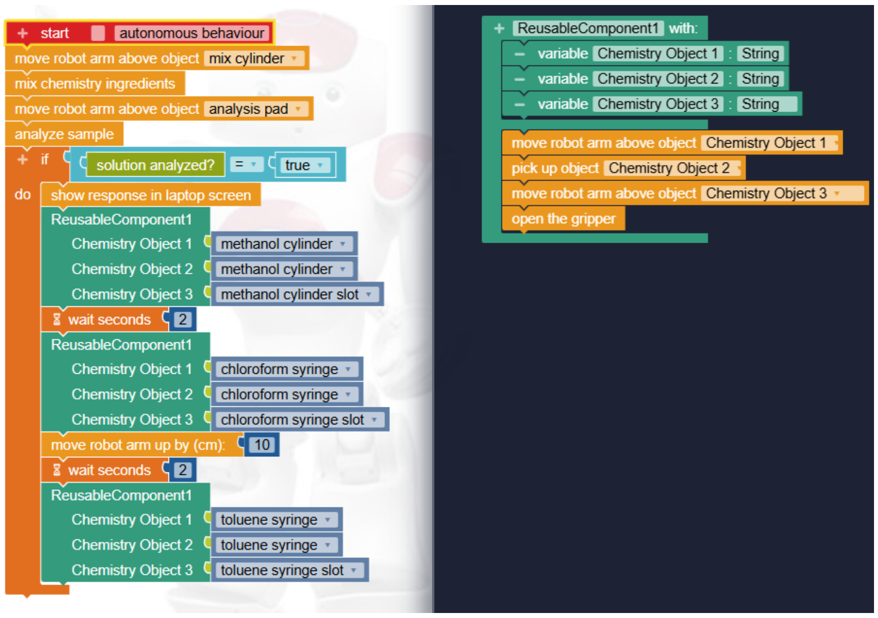
Ethical Considerations and Sampling. Prior to the commencement of the study, all participants are required to sign a consent form acknowledging their voluntary participation and granting permission for screen recording and data usage. It should be noted that this recruitment strategy constitutes *convenience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

3.3.2 Task Execution. The participants were initially given a short introduction to the OpenRoberta UI, as well as the mujoco robot simulator. They then performed 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 were 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. 5. The optimal solution implemented in OpenRoberta, utilizing a custom block for the object placement sequence.

Instead of creating a long linear sequence of blocks, 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:** Measured for both versions (Enhanced and Original) to compare performance across groups. Statistical analysis included paired t-tests to evaluate within-group improvements and between-group comparisons to identify order effects (carryover effects).
(2) **Solution Accuracy:** Evaluated by comparing participant solutions against the optimal reference solution. The primary metric was the voluntary adoption of reusable custom blocks, with assessment of whether participants successfully implemented code reuse practices or relied on linear, repetitive code structures.
(3) **Usability Assessment:** Evaluated using the System Usability Scale (SUS) questionnaire to measure participants' perceived usability of the Reuse Assistant feature.

(4) **Workload Assessment:** Measured using the NASA Task Load Index (NASA-TLX) to assess the cognitive demands imposed by the Reuse Assistant across six dimensions (mental demand, physical demand, temporal demand, performance, effort, and frustration).

This comprehensive evaluation provided a detailed understanding of how useful and effective is the Reuse Assistant feature to the end-users.

4 Results

4.1 Research Question 1: Can the Reuse Assistant improve the end-users performance?

Participant ID	Group (Order)	Completion time (Enhanced)	Completion time (Original)	Time Differe
P01	A (Enhanced → Original)	481 seconds	331 seconds	150 sec
P03	A (Enhanced → Original)	921 seconds	275 seconds	646 sec
P06	A (Enhanced → Original)	733 seconds	314 seconds	419 sec
P07	A (Enhanced → Original)	437 seconds	296 seconds	141 sec
P09	A (Enhanced → Original)	453 seconds	348 seconds	105 sec
P02	B (Original → Enhanced)	411 seconds	477 seconds	-66 sec
P04	B (Original → Enhanced)	189 seconds	435 seconds	-246 sec
P05	B (Original → Enhanced)	200 seconds	367 seconds	-167 sec
P08	B (Original → Enhanced)	266 seconds	485 seconds	-219 sec
P10	B (Original → Enhanced)	259 seconds	506 seconds	-247 sec

Table 2. Comparison of Time Taken by Participants in Groups A and B

Test	t-Value	p-value
Overall Comparison	-0.54	.602
Group A Improvement	-2.79	.049
Group B Improvement	5.56	.005
Carryover Effect	-4.37	.008

Table 3. Statistical Test Results

4.1.1 *Statistical Analysis.*

To evaluate the effectiveness of the Enhanced version, we conducted paired t-tests on the completion times. A paired t-test is a parametric statistical method used to compare two related samples. In this case, the same participants' performance under two different conditions (Enhanced vs. Original versions of OpenRoberta). This test is appropriate for our crossover design because each participant serves as their own control, reducing variability from individual differences. The paired t-test assesses whether the mean difference between paired observations is statistically significant, making it ideal for detecting within-subject changes in task completion time. We chose this method over independent t-tests because our data are not independent. Each participant contributed two measurements under different experimental conditions.

The analysis reveals distinct patterns between groups and identifies a significant order effect.

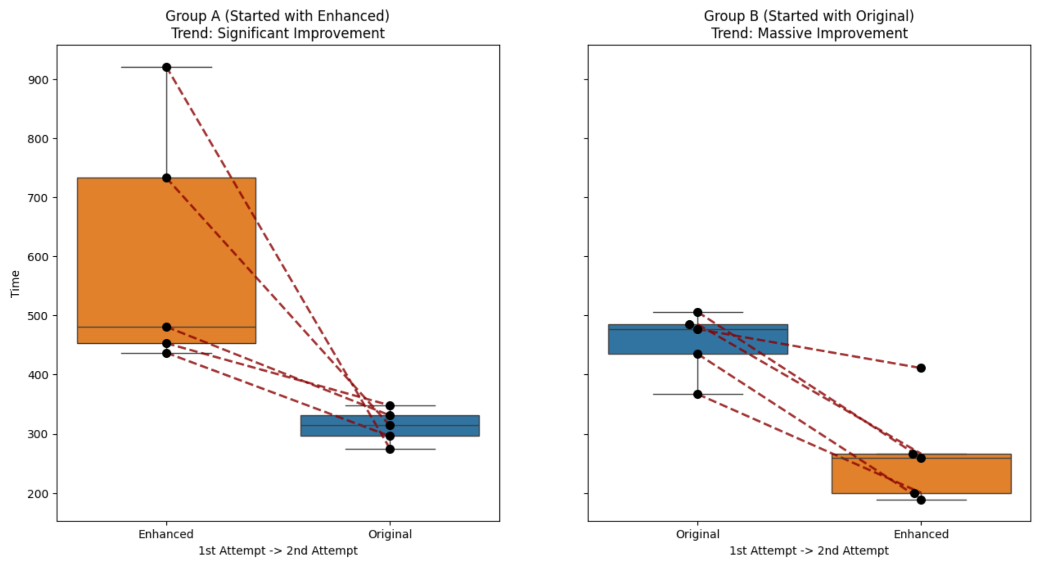


Fig. 6. Boxplot of task completion times

Overall Comparison. When combining all 10 participants regardless of order, the overall mean difference was -51.60 seconds (Standard Deviation = 302.10), yielding t = -0.54 (df = 9, p = .602). This non-significant result indicates no overall difference when order is ignored, highlighting the critical importance of presentation sequence.

Group B Analysis (Original → Enhanced). Group B participants started with the Original version and then used the Enhanced version. We calculated the difference (Original - Enhanced) for each participant, with positive values indicating faster performance on Enhanced. The mean improvement was 189.0 seconds (SD = 76.04), yielding a t-value of 5.558 (df = 4, p = .005). This statistically significant result demonstrates that participants who learned with the Original version first showed substantial speed improvements when switching to the Enhanced version.

Group A Analysis (Enhanced → Original). Group A started with Enhanced and switched to Original. The mean difference (Original - Enhanced) was -292.2 seconds (Standard Deviation = 234.19), producing a t-value of -2.79 (df = 4, p = .049). The negative value indicates that these participants were *slower* on the Enhanced version when it was presented first. This counterintuitive finding suggests a learning curve effect: participants encountered the automated reuse features before developing manual strategies, potentially requiring more time to understand the tool's suggestions.

Order Effect (Carryover Effect). To determine whether the order in which participants experienced the two versions influenced their performance, we conducted a Welch's t-test comparing the improvement scores between Group A and Group B. This analysis revealed a highly significant order effect (t = -4.37, df ≈ 5, p = .008).

The magnitude of this effect is substantial: there was a gap of -481.2 seconds between the two groups' mean improvement scores. Group B participants, who started with the Original version, showed an average improvement of +189.0 seconds when they switched to Enhanced. In contrast, Group A participants, who started with Enhanced, showed an average change of -292.2 seconds (meaning they were actually slower on Enhanced). This creates a total difference of approximately 481 seconds between the groups' experiences.

This finding demonstrates that presentation order profoundly impacts user performance. Participants who first struggled with the original version (Group B) were able to recognize and appreciate the value of the automated reuse feature when they encountered them second. Conversely, participants who received automated assistance immediately (Group A) had not yet developed the mental model of manual block assembly, making it harder for them to understand what the tool was helping them avoid. This suggests that prior experience with manual coding strategies is crucial for users to fully appreciate and effectively utilize automated assistance features.

4.1.2 *Solution Accuracy.*

Solution accuracy was evaluated by comparing participant solutions against the optimal reference solution defined in the treatment validation (see Section 3.3).

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.

While some participants provided unique solutions that differed slightly from the optimal reference solution such as skipping certain precautionary steps or reordering non-critical operations, these variations were deemed acceptable. The differences primarily reflected domain-specific safety practices that would matter in a real chemistry laboratory environment but had no impact on the robot simulator's execution behavior. Since the simulator performed identically regardless of these variations, all solutions were considered functionally correct.

4.2 Research Question 2: Is the Reuse Assistant assessed as sufficiently usable for the end-users?

To answer the second research question regarding the perceived usability of the system, we administered the System Usability Scale (SUS) questionnaire to all $N = 10$ participants immediately following the treatment validation.

The SUS yields a single number representing a composite measure of the overall usability of the system, with scores ranging from 0 to 100.

4.2.1 *Overall Usability Scores.*

The analysis of the survey data yielded a mean SUS score of **84.1** (*Median* = 80.0). According to the interpretive ranges defined by Bangor et al., a score above 80.3 is considered "Excellent" and places the system in the top 10% of products in terms of usability.

As detailed in Table 4, the individual scores ranged from a low of 52.5 to a perfect score of 100. Notably, 90% of participants (9 out of 10) rated the system above the industry average of 68, with the majority falling into the "Excellent" or "Very Good" categories.

4.2.2 *Distribution Analysis.*

The SUS scores demonstrate a strong positive skew, with 9 out of 10 participants (91%) rating the system above the industry average of 68. The distribution reveals tight clustering in two distinct bands: five participants (45%) achieved scores in the "Excellent" range (92.5-100), while another five participants (45%) scored in the "Very Good" range (75-80). This bimodal clustering pattern, with no scores between 52.5 and 75, suggests that users either readily adopted the system's paradigm or encountered initial conceptual barriers.

The score range spans 47.5 points (52.5 to 100), with the median (80.0) slightly below the mean (84.1), indicating that high-scoring participants pull the average upward. The consistency among the top 10 participants is particularly noteworthy, with all scores falling within a 25-point band, demonstrating reliable usability for the vast majority of the

Participant ID	SUS Score	Adjective Rating
P08	100.0	Excellent
P02	97.5	Excellent
P04	95.0	Excellent
P10	95.0	Excellent
P01	92.5	Excellent
P07	80.0	Very Good
P06	80.0	Very Good
P05	77.5	Very Good
P09	75.0	Very Good
P03	52.5	OK
Mean Score	**84.5**	**Excellent**

Table 4. Individual System Usability Scale (SUS) Scores

target user population. The single outlier at 52.5, while representing only 9% of participants, warrants investigation as it may identify a specific user profile or interaction pattern requiring additional support.

One outlier was observed (Participant 3, Score: 52.5), who indicated needing technical support and additional learning time. This suggests a slight learning curve for users less comfortable with block-based encapsulation concepts, though as an isolated case, it does not negate the general consensus of high usability.

4.3 Research Question 3: What is the perceived workload when using the Reuse Assistant?

To assess the cognitive demands imposed by the Reuse Assistant, we administered the NASA Task Load Index (NASA-TLX) questionnaire to all participants after completing the task with the enhanced version. The NASA-TLX is a widely used multidimensional assessment tool that measures perceived workload across six subscales, each rated on a scale from 1 (Very Low) to 10 (Very High).

4.3.1 Overall Workload Assessment. The analysis of NASA-TLX data yielded an overall mean workload score of **1.88** across all six dimensions. According to NASA-TLX interpretation guidelines, scores below 2.0 indicate very low perceived workload, suggesting that the Reuse Assistant imposed minimal cognitive and physical burden on users.

4.3.2 Dimension-Specific Analysis. As shown in Table 5, the six NASA-TLX dimensions assessed were:

- **Mental Demand:** How much mental and perceptual activity was required to understand and use the Reuse Assistant? (Mean: 2.2)
- **Physical Demand:** How much physical activity was required (e.g., clicking, modifying program) while using the Reuse Assistant? (Mean: 1.9)
- **Temporal Demand:** How much time pressure did you feel while completing the task using the Reuse Assistant? (Mean: 1.2)
- **Performance:** How successful do you think you were in accomplishing the goals using the Reuse Assistant? (Mean: 1.9)
- **Effort:** How hard did you have to work to accomplish your level of performance when using the Reuse Assistant? (Mean: 2.8)

Participant	Mental	Physical	Temporal	Performance	Effort	Frustration	Mean
P01	2	1	1	1	2	1	1.33
P02	1	1	1	1	1	1	1.00
P03	5	5	1	5	5	5	4.33
P04	1	1	1	1	2	1	1.17
P05	3	3	1	2	3	2	2.33
P06	4	2	1	3	5	2	2.83
P07	2	1	1	2	3	1	1.67
P08	1	1	1	1	1	1	1.00
P09	2	2	1	2	2	2	1.83
P10	1	1	2	1	2	1	1.33
						Overall Mean Score:	**1.88**

Table 5. NASA-TLX Workload Scores by Participant and Dimension

- **Frustration:** How insecure, discouraged, irritated, stressed and annoyed were you when using the Reuse Assistant feature? (Mean: 1.7)

The temporal demand dimension received the lowest mean rating (1.2), indicating that participants experienced minimal time pressure. The effort dimension received the highest rating (2.8), though still well within the low workload range, suggesting that while some concentration was required, it remained manageable for most users.

4.3.3 Outlier Consideration. Consistent with the SUS findings, Participant P03 reported significantly elevated workload across most dimensions (mean: 4.33), with ratings of 5 (High) for mental demand, physical demand, effort, and frustration. This participant corresponds to the same individual who reported the lowest SUS score (52.5), reinforcing the pattern that a small subset of users may require additional support or training to effectively utilize the system's automation features.

Excluding this outlier, the remaining nine participants showed remarkably consistent low workload scores (mean: 1.58), with eight of nine reporting individual mean scores below 2.0. This consistency demonstrates that the Reuse Assistant successfully minimizes cognitive burden for the vast majority of the target user population.

5 Discussion

This study evaluated the Reuse Assistant, an automated guidance tool designed to help end-users recognize and implement code reuse in block-based programming environments. Through a crossover study with 10 participants from the chemistry domain, we assessed the tool's impact on performance (RQ1), usability (RQ2), and perceived workload (RQ3). The findings reveal both the potential and limitations of automated assistance in promoting software reuse practices among domain experts with limited programming expertise.

5.1 Implications for Theory

5.1.1 Reducing Attention Investment through Proactive Assistance. Our study provides empirical support for the Attention Investment Model [3] as applied to end-user development tools. The Attention Investment Model states that users make rational decisions about whether to adopt new tools or features based on a cost-benefit analysis of the attention they must invest upfront versus the perceived benefits they expect to receive [4]. The higher the upfront attention cost (learning curve, discovery effort, comprehension requirements), the less likely users are to adopt and utilize available features, even when those features would ultimately benefit their work.

In the standard OpenRoberta environment, creating reusable custom blocks requires users to: (1) recognize that code duplication exists and represents an opportunity for abstraction, (2) discover that custom block functionality is available in the system, (3) locate where this feature resides in the interface, (4) understand how to use the feature correctly, and (5) manually configure the custom block with appropriate parameters. This multi-step process represents a large upfront attention investment that end-users, focused on their primary domain tasks rather than software engineering practices, are unwilling or unable to make. The result is 0% adoption of reusable components in our standard condition, despite participants possessing the cognitive capacity to understand and use custom blocks when guided to do so.

The Reuse Assistant changes this attention investment equation by eliminating steps 1-4 entirely. Users do not need to recognize duplication patterns (the system detects them automatically), discover the feature exists (the system actively presents opportunities), locate the feature in the interface (visual highlighting brings the opportunity directly to users' attention), or understand complex configuration procedures (automated parameterization handles technical details). The upfront attention investment is reduced to a single decision: accept or reject the system's suggestion. This dramatic reduction in cognitive cost (from a complex multi-step learning process to a binary choice) explains the 100% adoption rate in the Enhanced condition.

Our findings extend the Attention Investment Model [3] by demonstrating that proactive, context-aware assistance can transform feature adoption from an investment decision into an opportunistic choice. Rather than requiring users to invest attention before experiencing any benefit, the Reuse Assistant delivers immediate, real value (highlighted duplicates, one-click refactoring) that users can evaluate in real-time within their workflow. This "zero-cost trial" approach eliminates the adoption barrier built-in to traditional feature-discovery models, where users must commit attention resources before knowing whether the investment will prove worthwhile [4].

5.1.2 Addressing the Selection Barrier in End-User Development. The results provide empirical evidence for a critical distinction between different types of barriers to software reuse within Ko et al.'s [6] learning barriers framework. Among the six barriers identified by Ko and colleagues (design, selection, coordination, use, understanding, and information barriers), our work specifically addresses the *selection barrier*: the difficulty users face in knowing where to look for features and choosing appropriate tools from the available options.

The selection barrier appears in two distinct ways in block-based programming environments [6]. First, users must know that reuse mechanisms exist and where to find them within the interface. Second, even when aware of available features, users must determine when and how to apply them appropriately. Our 0% adoption rate in the standard OpenRoberta condition demonstrates that the selection barrier is impossible to overcome for domain experts without programming backgrounds, even when the interface provides the necessary functionality. Participants did not lack the capability to create custom blocks (the same individuals achieved 100% adoption in the Enhanced condition) but rather lacked the knowledge of where to look for this feature and when to apply it.

The Reuse Assistant eliminates the selection barrier through two supporting mechanisms. First, automated detection makes the feature location irrelevant. Users do not need to search the interface because the system proactively brings the functionality to their attention at the appropriate moment. Second, context-aware suggestions eliminate the decision burden about when to apply reuse. The system identifies appropriate opportunities and presents them when relevant, allowing users to focus on domain-level acceptance decisions rather than technical feature selection.

This finding extends Ko et al.'s framework [6] by demonstrating that in block-based environments targeting end-users, the selection barrier comes before and is more important than other barriers. The low NASA-TLX workload scores (mean: 1.88) and high SUS scores (mean: 84.1) indicate that once the selection barrier is removed, once users no longer need to find and choose features, the remaining barriers (use, understanding, coordination) impose minimal mental burden. This suggests that tool designers should prioritize eliminating selection barriers through proactive assistance before addressing other barrier types, as the latter become manageable once users are successfully guided to appropriate features.

Relationship Between Recognition and Selection Barriers. While Ko et al.'s [6] selection barrier focuses on knowing where to look for features, our work identifies a related but distinct *recognition barrier* specific to code reuse: users' inability to identify opportunities for abstraction within their own code. These barriers are complementary. Even if users know where the custom block feature is located (overcoming the selection barrier), they cannot use it effectively without recognizing when their code contains patterns suitable for abstraction (overcoming the recognition barrier). Our Reuse Assistant addresses both barriers simultaneously through automated pattern detection (recognition) and proactive presentation (selection), explaining the dramatic shift from 0% to 100% adoption.

5.1.3 The Order Effect: Prior Experience as a Prerequisite for Appreciating Automation. The significant order effect (t=-4.37, p=.008) reveals a counter-intuitive finding: participants who received automated assistance first were actually *slower* to complete tasks than those who first struggled with the manual approach. This 481-second performance gap suggests that automation effectiveness depends on users having established mental models of the problem space.

This finding has theoretical implications for understanding how end-users learn to value productivity tools. Participants in Group B (Original → Enhanced) developed an experiential baseline that allowed them to recognize what the automation was helping them avoid. In contrast, Group A participants (Enhanced → Original) lacked this reference frame, potentially viewing the automated suggestions as interruptions rather than assistance.

This aligns with theories of learning transfer and expertise development [6], suggesting that some exposure to manual processes may be valuable for teaching before introducing automation. It challenges the assumption that "easier is always better" in tool design, indicating that mental struggle during initial learning may enhance appreciation and effective use of advanced features.

5.2 Implications for Practice

5.2.1 High Usability and Low Workload Support Simple Design Principles. The SUS results (mean: 84.1) place the Reuse Assistant in the "Excellent" category, with 90% of participants rating it above the industry average of 68. This high usability score demonstrates that automated guidance can be both powerful and easy to use. The feature achieved this by focusing on simplicity: visual highlighting to show duplicates and one-click acceptance to create reusable blocks. This suggests that effective end-user tools should prioritize clarity over complexity.

The NASA-TLX results (mean: 1.88, with temporal demand at 1.2 and frustration at 1.7) further support this finding, showing that effective guidance does not require complex interactions. The combination of high SUS scores and low NASA-TLX scores demonstrates that the Reuse Assistant successfully reduces barriers without adding mental burden. The key to this success is *making the invisible visible* rather than increasing system sophistication.

The bimodal distribution in both SUS scores and NASA-TLX workload (with one consistent outlier) suggests that while most users experience minimal burden, a small subset encounters significant difficulties. This pattern indicates individual differences in openness to automated guidance, potentially related to prior mental models, learning preferences, or comfort with system-initiated interactions.

5.2.2 Design Principle 1: From Passive Toolboxes to Active Assistants. Current block-based programming environments (Scratch, Blockly, standard OpenRoberta) follow a passive interaction model where reuse mechanisms exist as features waiting to be discovered. Our 0% adoption rate in the standard condition demonstrates the limitations of this approach for end-user developers, as participants created functional but non-optimal solutions using linear, repetitive code structures. The 100% adoption rate with automated detection proves that tool designers must shift from providing capabilities to actively guiding their use.

Practical Recommendation: Development environments targeting domain experts should implement background analysis systems that continuously monitor for patterns that show code smells (repetition, long sequences, similar structures). Rather than requiring users to manually invoke refactoring tools, the system should actively show opportunities through non-intrusive notifications. This "ambient intelligence" approach respects user agency (through opt-in confirmations) while addressing the fundamental recognition barrier.

5.2.3 Design Principle 2: Strategic Introduction of Automation. The order effect findings have direct implications for training and onboarding. Organizations introducing automated coding assistants should consider implementing a staged approach:

(1) **Initial Exposure Phase:** Allow users to complete initial tasks without automated assistance, building experiential understanding of manual processes and their pain points.
(2) **Guided Automation Phase:** Introduce automated suggestions after users have established baseline workflows, ensuring they can appreciate what the automation provides.
(3) **Full Automation Phase:** Enable all automation features once users have developed adequate mental models.

This staged approach goes against the intuitive "make it easy from the start" philosophy but may lead to better long-term adoption and appropriate use of automation features.

5.2.4 Design Principle 3: Minimize Interaction Complexity. The exceptionally low NASA-TLX scores (temporal demand: 1.2, frustration: 1.7) demonstrate that effective guidance need not be complex. The Reuse Assistant succeeded through:

- **Visual highlighting:** Simple color change to indicate detected patterns
- **One-click acceptance:** Single confirmation to trigger automated refactoring
- **Immediate feedback:** Instant display of the created reusable component

Practical Recommendation: Designers should resist the temptation to add configuration options, customization parameters, or complex workflows to guidance features. The key is *making the invisible visible*, not providing sophisticated controls. For end-user developers, the interaction cost must be minimal to avoid creating new barriers while removing old ones.

5.2.5 Design Principle 4: Plan for Individual Differences. The consistent outlier pattern (one participant with low SUS scores and high workload across all dimensions) indicates that approximately 10% of users may struggle with automated guidance. This is likely cannot be avoided given individual differences in learning preferences and comfort with system-initiated interactions.

Practical Recommendation: Provide a clearly visible mechanism to disable automated suggestions for users who find them distracting or confusing. Additionally, supplement automated detection with alternative pathways (manual invocation, documentation, tutorials) to ensure users who reject proactive guidance can still access reuse mechanisms if they choose to seek them out.

5.3 Threats to Validity

5.3.1 Internal Validity.

Carryover effect. While the crossover design allowed within-subjects comparison, the significant order effect (p=.008) indicates that the sequence of conditions fundamentally altered the user experience. This carryover effect means we cannot cleanly separate the impact of the Reuse Assistant from the impact of prior experience. The 481-second performance gap between groups suggests that learning from the first condition substantially influenced performance in the second condition.

Mitigation: We explicitly analyzed and reported the order effect as a finding rather than treating it as unwanted noise. The crossover design, despite this limitation, provided valuable insights about how prior experience shapes users' ability to benefit from automation. Future studies could employ between-subjects designs to isolate tool effects, though this would sacrifice statistical power given small sample sizes typical in EUD research.

5.3.2 External Validity.

Convenience Sampling and Population Representation. Participants were recruited through the researchers' professional networks at the University of Southern Denmark, creating a convenience sample. This introduces several limitations:

- **Geographic and institutional diversity:** While the study included participants from multiple countries (both local and international participants connected online), recruitment relied primarily on the researchers' professional networks, which may not represent the full geographic and cultural diversity of potential end-users in laboratory automation contexts.
- **Self-selection bias:** Volunteers may be more technologically inclined or motivated than typical professionals who would use cobot programming in practice, potentially overestimating the tool's ease of adoption among less motivated users.
- **Domain representation:** While participants came from diverse scientific backgrounds (chemistry, agronomy, biochemistry) united by laboratory coursework experience, they represent primarily academic contexts rather than industrial laboratory settings where cobot programming would be used professionally.
- **Sample size:** With N=10 for performance evaluation,usability assessment and workload assessment, the study lacks statistical power to detect small effects or to adequately characterize rare user profiles (such as the consistent outlier), limiting the generalizability of findings to broader populations.

Implications: Findings should be interpreted as preliminary evidence rather than final proof of effectiveness across all end-user developer populations. Replication studies with larger, more diverse samples from multiple institutions and countries are necessary to establish the robustness of these results.

Ecological Validity: Laboratory vs. Authentic Use. The study was conducted in a controlled setting with researcher guidance available, tasks completed in a single session, and no real-world consequences for errors. This differs from authentic usage where:

- Users work independently without expert support
- Programming tasks span multiple sessions with interruptions

- Errors in cobot programs could damage equipment or compromise experiments
- Users balance programming with their primary professional responsibilities

Mitigation: We included chemistry domain experts as participants rather than generic users, and the task was based on actual laboratory procedures. However, longitudinal field studies observing the Reuse Assistant in authentic work contexts are necessary to validate its practical impact.

5.3.3 Construct Validity.

Measurement Instruments. We used standardized instruments (SUS, NASA-TLX) which have established validity in usability research. However:

- **SUS limitation:** Measures perceived usability rather than objective usability metrics such as error rates or task success beyond completion time.
- **NASA-TLX limitation:** Assesses subjective workload perception, which may not correlate perfectly with objective cognitive load or learning outcomes.
- **Performance metrics:** Completion time captures efficiency but not code quality, maintainability, or the user's conceptual understanding of reuse principles.

Single Outlier Pattern. One participant (P03) consistently reported low usability (SUS: 52.5) and high workload (NASA-TLX: 4.33) across all measures. While we interpreted this as evidence of individual differences, alternative explanations include:

- Technical issues during the session (software bugs, hardware problems)
- Misunderstanding of questionnaire items or rating scales
- Fatigue or external stressors unrelated to the tool
- Genuine fundamental incompatibility between the user's mental model and the tool's interaction paradigm

Limitation: With only one outlier, we cannot determine which explanation is correct or whether this represents 10% of the population or a unique case. Larger samples are needed to characterize the distribution of user experiences.

6 Conclusion and Future Work

This study examined whether automated guidance can help end-users recognize and apply code reuse in block-based programming. We developed the Reuse Assistant, a tool that automatically detects duplicate code sequences and guides users to create reusable custom blocks in the OpenRoberta environment.

The results showed a clear difference in reuse adoption. While no participants created reusable blocks in the standard environment, all participants successfully used them with the Reuse Assistant. The tool received high usability ratings (SUS mean: 84.1) and low workload scores (NASA-TLX mean: 1.88), demonstrating that automated guidance can be both effective and easy to use.

Our findings contribute to theory by extending the Attention Investment Model and the Learning Barriers Framework. We showed that proactive assistance reduces the upfront cost of adopting new features and that the selection barrier is particularly important in block-based environments for end-users. The significant order effect revealed that prior manual experience helps users appreciate automation benefits.

For practice, this study demonstrates that simple design choices matter. Visual highlighting, one-click acceptance, and immediate feedback were sufficient to achieve high adoption without adding complexity. The results suggest that programming environments for domain experts should actively guide users rather than waiting for them to discover features independently.

6.1 Future Work

Future research should address several limitations of this study. First, longitudinal studies in authentic work settings would reveal whether users internalize reuse concepts or remain dependent on automated detection. Second, studies with larger and more diverse samples would better characterize individual differences in receptivity to automated guidance. Third, the tool should be tested with more complex tasks featuring less obvious duplication patterns and nested structures. Finally, research should explore whether the approach generalizes to other end-user programming contexts beyond block-based robotics, such as spreadsheet programming or workflow automation.

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] Alan F. Blackwell. 2002. First Steps in Programming: A Rationale for Attention Investment Models. In *Proceedings of the IEEE Symposia on Human Centric Computing Languages and Environments*. IEEE Computer Society, 2–10. doi:10.1109/HCC.2002.1046334

[4] Alan F. Blackwell and Thomas R. G. Green. 2003. Notational Systems - The Cognitive Dimensions of Notations Framework. *HCI Models, Theories, and Frameworks: Toward a Multidisciplinary Science* (2003), 103–133.

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

[6] Andrew J. Ko, Brad A. Myers, and Htet Htet Aung. 2004. Six Learning Barriers in End-User Programming Systems. *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)* (2004), 199–206. doi:10.1109/VLHCC.2004.47

[7] 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

[8] 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

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

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

A System Usability Scale (SUS) Questionnaire

The System Usability Scale (SUS) is a widely used standardized questionnaire for assessing the perceived usability of a system. Participants respond to each statement using a 5-point Likert scale ranging from 1 (Strongly Disagree) to 5 (Strongly Agree). The SUS score is calculated by converting the responses to a scale of 0-100, where higher scores indicate better usability.

A.1 SUS Statements

(1) I think that I would like to use the Reuse Assistant feature frequently.
(2) I found the Reuse Assistant feature unnecessarily complex.
(3) I thought the Reuse Assistant feature was easy to use.
(4) I think that I would need the support of a technical person to be able to use the Reuse Assistant feature.
(5) I found the various functions in the Reuse Assistant feature were well integrated.
(6) I thought there was too much inconsistency in the Reuse Assistant feature.
(7) I would imagine that most people would learn to use the Reuse Assistant feature very quickly.
(8) I found the Reuse Assistant feature very cumbersome to use.

(9) I felt very confident using the Reuse Assistant feature.
(10) I needed to learn a lot of things before I could get going with the Reuse Assistant feature.

A.2 Scoring Method

For odd-numbered items (1, 3, 5, 7, 9), the score contribution is the scale position minus 1. For even-numbered items (2, 4, 6, 8, 10), the contribution is 5 minus the scale position. The sum of all item contributions is then multiplied by 2.5 to obtain the overall SUS score, which ranges from 0 to 100.