Guiding End-Users towards 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*—End-users who program collaborative robots for laboratory automation often create repetitive code because they struggle to recognize opportunities for reuse. ot actWhile block-based programming environments provide accessible interfaces, they do not actively guide end-users toward creating reusable components. 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 within the OpenRoberta environment and guides users to create reusable custom blocks through visual highlighting and one-click refactoring. Through a within-subjects study with 18 participants from the chemistry domain, we evaluated the feature's impact on performance, usability, and perceived workload. Automated guidance increased reuse adoption from 0% in the standard OpenRoberta version to 100% when using the Reuse Assistant. The feature achieved high usability scores (SUS mean: 84.03) and imposed minimal cognitive burden (NASA-TLX mean score: 1.92). The significant order effect revealed that prior manual experience helps users appreciate automation benefits. This dramatic shift in adoption suggests that end-users are capable of using advanced features if the system actively guides them.**

CCS Concepts: • **Software and its engineering** → **Reusability**.

ACM Reference Format:
Anne-Marie Rommerdahl, Jeremy Alexander Ramírez Galeotti, Dimitrios Dafnis, Nasifa Akter, and Mohammad Hosein Kardouni. 2018. Guiding End-Users towards Software Reuse: An Evaluation of Automated Assistance in Block-Based Programming for Chemistry Laboratory Automation. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation email (Conference acronym 'XX).* ACM, New York, NY, USA, 23 pages. https://doi.org/XXXXXXX.XXXXXXX

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

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.

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. Through a within-subjects study with 18 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 four research questions:

1) How does the Reuse Assistant affect the end-users' performance?
2) To what extent does the reuse assistant facilitate reusability?
3) How do the end-users assess the reuse assistant in terms of usability?
4) How do the end-users assess the perceived workload when operating the Reuse Assistant?

The results showed that automated guidance increased reuse adoption from 0% to 100%, achieved high usability scores (SUS mean: 84.03), and imposed minimal cognitive burden (NASA-TLX mean score: 2).

The contributions of this work are both theoretical and practical. We extend the Attention Investment Model [4] and Learning Barriers Framework [10] by demonstrating that proactive assistance transforms feature adoption from a high-cost investment to a low-cost opportunistic choice, effectively reducing the selection barrier.

2 Background and Related Work

Software reuse is the practice of reusing previously written code, rather than writing new code 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'[3], 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 also design their own reusable components, keeping all the logic in one place, making both testing and maintenance easier to perform. 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 by Bock and Frank (2021) studied several low-code platforms (LCPs), in order to identify characteristic features of LCPs. The identified features were presented according to how frequently 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[6]. 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, such as abstraction or reuse, can enhance learning outcomes[11].

There have been proposed some ideas on how to promote reuse for LCPs, such as the templating language OSTRICH, developed for the model-driven low-code platform OutSystems[12]. 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[13], 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[14] 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, well-known pitfalls of AI are its tendency to generate non-deterministic outputs, and hallucinations. While an experienced programmer can critically analyze the output of the AI, the common end-user lacks that ability. 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[1]. Their findings demonstrated that automated refactoring can significantly enhance code quality and readability for novice programmers.

Building upon all these concepts and ideas, our project introduces a guided Reuse Assistant within the OpenRoberta Lab environment. The tool is designed to help users identify and apply reuse more easily while creating their robot programs. Focused on guiding users toward creating reusable custom blocks to promote modularity and abstraction, the tool automatically scans 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 can be reused. 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 [16], the 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, 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. 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, with the added bonus of not having to handle dangerous chemicals.

One critical challenge in EUD is code reuse. Users frequently create repetitive code as 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. So, while the use of robots in chemistry lab work offer great benefits, the challenge of automating the repetitive work may turn chemists away from using robots.

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

3.2 Treatment Design

To address the problem of code reuse in EUD for cobots, we have derived a set of requirements designed to contribute to the chemist's goal of creating maintainable and reusable robot programs. Functionally, the artifact must be capable of automatically detecting duplicate or similar block sequences and visually highlighting these duplications within the user's workspace. These requirements are necessary to help the end-user recognize opportunities for reuse, that would otherwise go unnoticed. Once detected, the system must offer to create 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. To satisfy the requirements above, we designed the Reuse Assistant as a feature for the Open Roberta Lab.

3.2.1 Architecture. The system enables the execution of block-based programs on a simulated cobot through a three-tier architecture, as illustrated in figure 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.

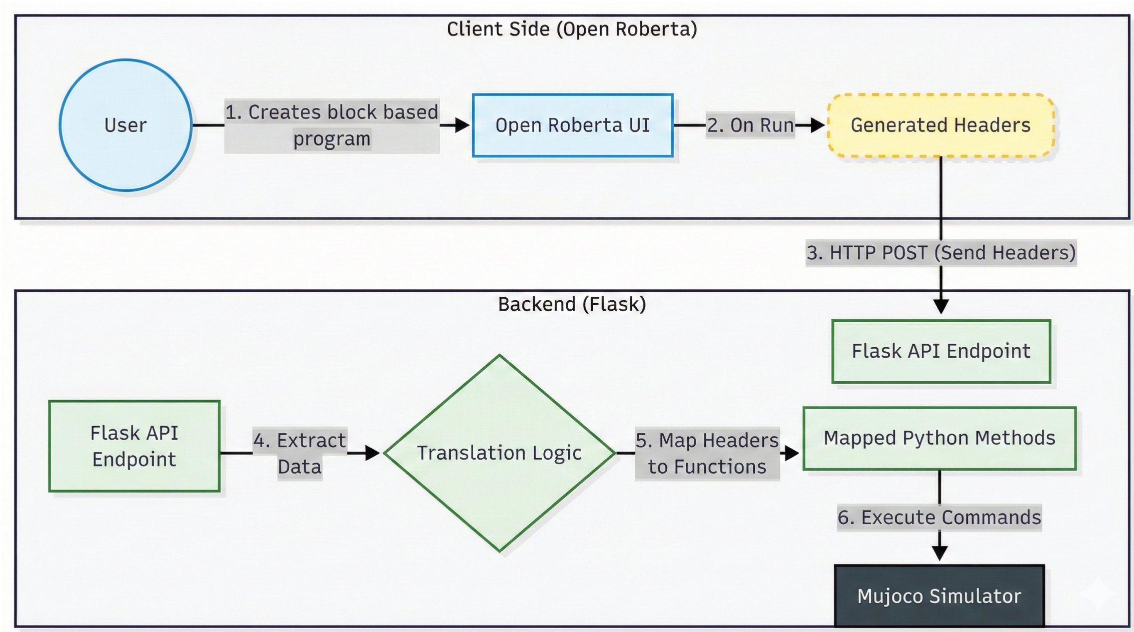

Fig. 1. System architecture: Data flow from Client Side to Simulator

3.2.2 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 which is illustrated in Algorithm 1 is short, explicit, and uses straightforward data structures (lists).

3.2.3 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.

Algorithm 1 Illustrates the core logic for identifying duplicate block sequences in the user's workspace.

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

```
Chain = buildLinearChain(StartBlock)
Sequences = List⟨sequence⟩
for startIndex = 0 to length(Chain) - 1 do
  for sequenceLength = 1 to MaxSequenceLength do
    sequence = Chain[startIndex .. startIndex + sequenceLength - 1]
    numberOfBlockTypesInSequence = getNumberOfDistinctBlockTypes(sequence)
    if sequenceLength >= MinimumSequenceLength and numberOfBlockTypesInSequence >= MinimumDifferentBlockTypesInSequence then
      Sequences.append(sequence) // record sequence occurrence
    end if
  end for
end for
ReusableComponentCandidates = {Sequences | occurrence ≥ 2}
sort ReusableComponentCandidates by (longest sequence length and most recent occurrence)
return ReusableComponentCandidates[0] // Return highest priority candidate
```

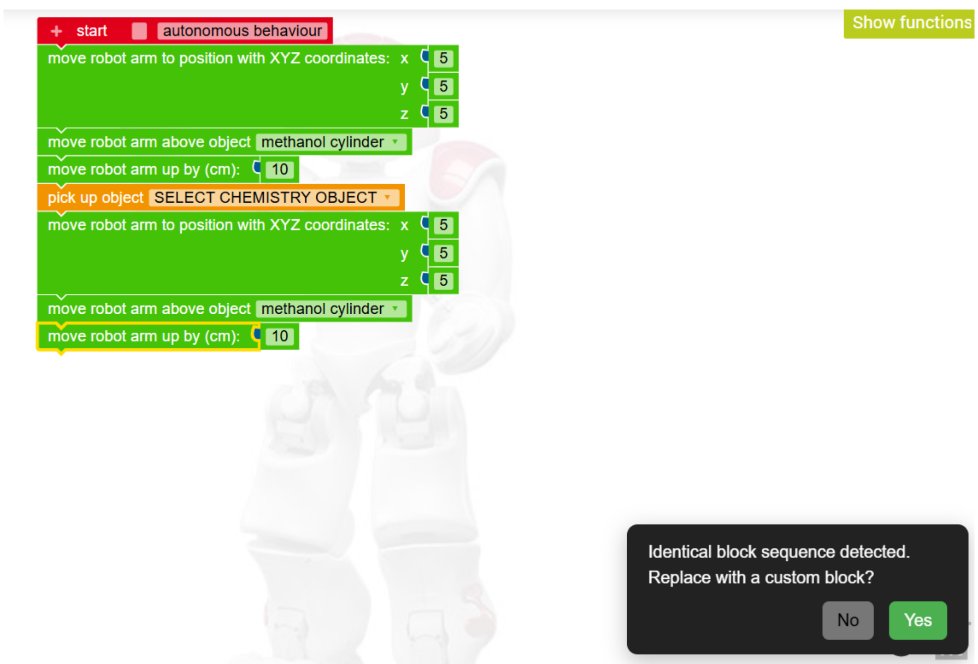

Fig. 2. Reuse Assistant workflow: detection - the interface detects and highlights duplicate blocks by changing their color to green.

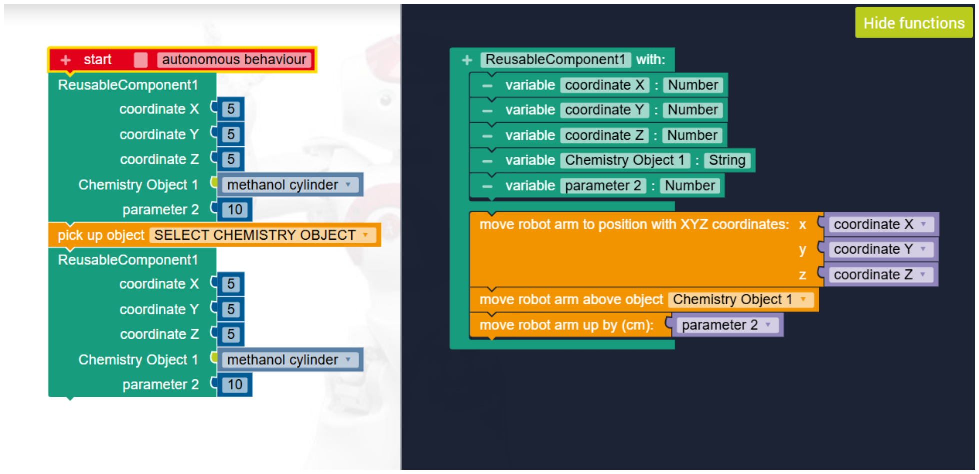

Fig. 3. Reuse Assistant workflow: refactoring - the automated refactoring result, showing the new custom block definition and the simplified main program.

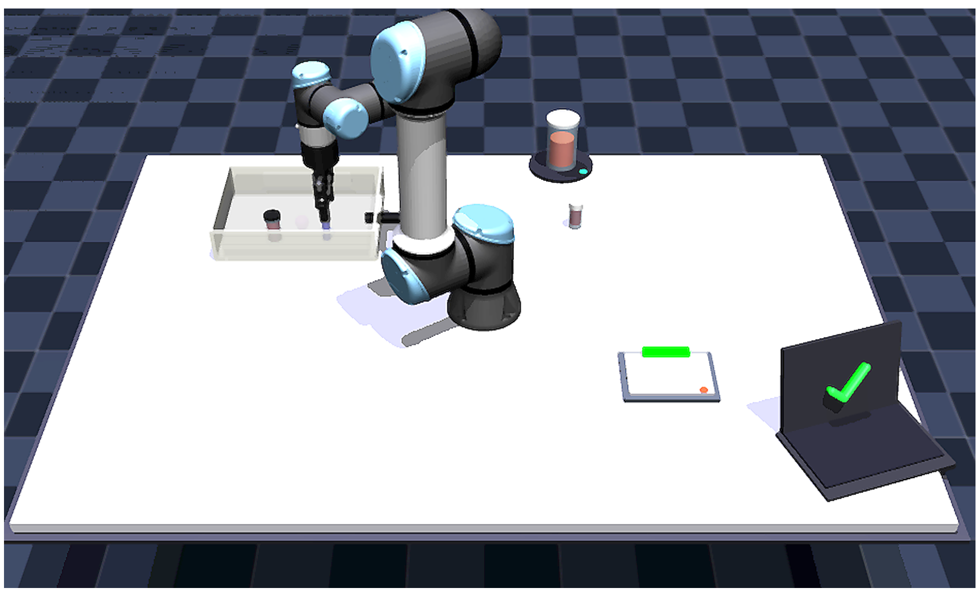

Fig. 4. Mujoco robot simulator executing the commands from Open Roberta.

3.3 Treatment Validation

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

3.3.1 Participant Recruitment. A total of 18 participants were selected with similar level of expertise in block-based programming. 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.

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 is illustrated in Figure 5. 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 enhanced version of OpenRoberta that includes the Reuse Assistant benefits and the original one that does not. Half of the participants will begin solving the task

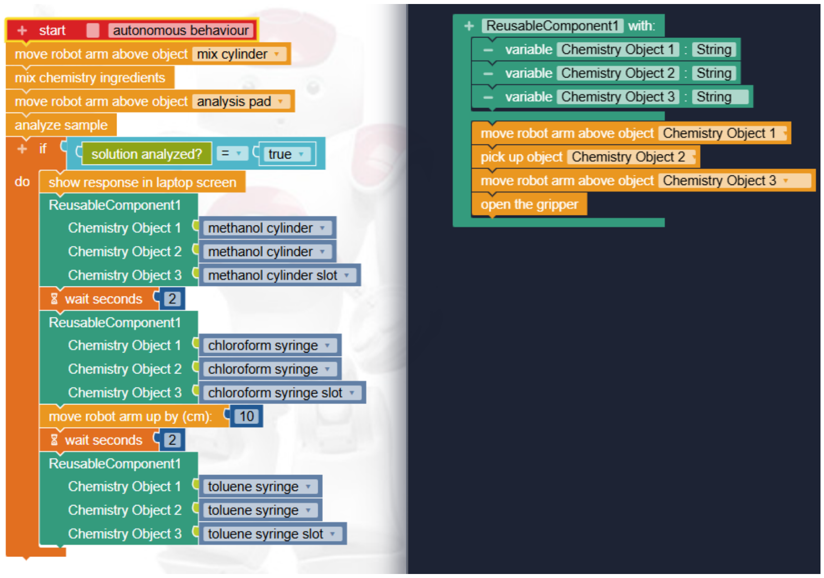

Fig. 5. The optimal solution implemented in OpenRoberta, utilizing a custom block for the object placement sequence.

first on the enhanced version, while the other half will start with the original 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 in seconds for both conditions (Enhanced and Original) to evaluate efficiency gains. Statistical analysis employed paired t-tests to assess within-group improvements and a Welch's t-test to compare improvement scores between groups, specifically isolating the impact of the order effect arising from the sequence of conditions.
(2) **Reuse adoption and functional correctness:** Evaluated by tracking the voluntary implementation of reusable custom blocks during the task. This primarily measured adoption rates (pass/fail of reuse implementation). Functional correctness was assessed by verifying if the robot successfully executed the chemical mixing protocol, with minor variations in safety steps deemed acceptable if they did not affect the simulation outcome.
(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 post-task Workload questionnaire (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: How does the Reuse Assistant affect the end-users' performance?

To evaluate the impact of the Reuse Assistant on end-user performance, we measured task completion times for all participants under both conditions (Enhanced and Original versions of OpenRoberta). The study employed a within subjects design where participants were divided into two groups: Group A experienced the Enhanced version of OpenRoberta first, while Group B started with the Original version. This design allowed us to assess not only the feature's effectiveness but also the potential order effect arising from the learning curve between the two conditions.

Tables 2 and 3 present the individual completion times for all participants across both conditions(with and without using the Reuse Assistant). The data reveal substantial variability in performance outcomes depending on the order in which the participants performed the tasks, with Group B participants showing consistent improvements when moving from the Original OpenRoberta version to the Enhanced, while Group A participants exhibited the opposite pattern.

Participant ID	Performance with Reuse Assistant	Performance without Reuse Assistant	Difference
P01	481 seconds	331 seconds	150 seconds
P03	515 seconds	320 seconds	195 seconds
P06	733 seconds	314 seconds	419 seconds
P07	437 seconds	296 seconds	141 seconds
P09	453 seconds	348 seconds	105 seconds
P11	735 seconds	364 seconds	371 seconds
P13	610 seconds	407 seconds	203 seconds
P15	410 seconds	540 seconds	-130 seconds
P17	560 seconds	440 seconds	120 seconds

Table 2. Task Completion Times of group A

Participant ID	Performance with Reuse Assistant	Performance without Reuse Assistant	Difference
P02	411 seconds	477 seconds	-66 seconds
P04	189 seconds	435 seconds	-246 seconds
P05	200 seconds	367 seconds	-167 seconds
P08	266 seconds	485 seconds	-219 seconds
P10	259 seconds	506 seconds	-247 seconds
P12	450 seconds	720 seconds	-270 seconds
P14	540 seconds	670 seconds	-130 seconds
P16	335 seconds	400 seconds	-65 seconds
P18	540 seconds	862 seconds	-322 seconds

Table 3. Task Completion Times of group B

The paired boxplots 6 illustrate the impact of the Reuse Assistant on task completion times across the two counter-balanced groups. Both groups demonstrated a learning effect, achieving faster times in their second attempt regardless of condition. However, the magnitude of improvement differed substantially between the groups. Group A, which transitioned from using the Reuse Assistant to working without it, showed an average time reduction of 174.9s (Standard Deviation = 158.9s). In contrast, Group B participants, which operated without the feature benefits in their first attempt and utilized the Reuse Assistant in their second attempt, exhibited a significantly larger efficiency gain, with an average time reduction of 192.4 seconds (Standard Deviation = 90.9 seconds). This suggests that while task familiarity contributed to speed, the introduction of the Reuse Assistant provided a distinct performance advantage.

To statistically evaluate these observed differences, we conducted paired t-tests [8] and a Welch's t-test [15] to compare performance improvements between groups. Table 4 summarizes the statistical test results, including overall comparisons, within-group improvements, and the analysis of the order effect.

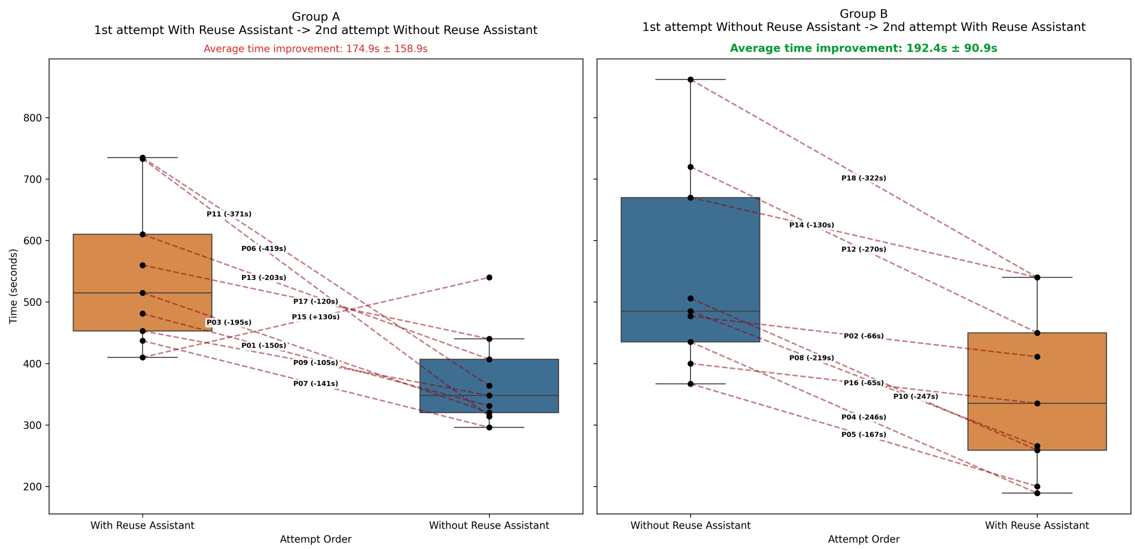

Fig. 6. Distribution of task completion times (in seconds) comparing Group A and Group B participants across both conditions (with and without Reuse Assistant).

Test	t-Value	p-value
Overall Comparison	-0.16	0.872
Group A Improvement	3.30	0.011
Group B Improvement	-6.35	< 0.001
Order Effect	6.02	< 0.001

Table 4. Statistical Test Results

4.1.1 *Performance statistical analysis.*

The analysis reveals distinct patterns between the two groups and identifies a significant carryover effect.

Overall Comparison. When combining all 18 participants regardless of the order in which they experienced the two OpenRoberta versions (the one with the reuse assistant feature benefits and the other without them), the overall mean time difference was -8.78 seconds (Standard Deviation = 226.90), leading to a t-value = -0.16 (p = 0.872). This non-significant result indicates no overall difference when order is ignored.

Group A Analysis. Group A participants experienced the benefits of the Reuse Assistant in their first attempt and then tried to solve the task without them in their second attempt. The mean difference (Completion time with Reuse Assistant - Completion time without Reuse Assistant) was 174.9 seconds (Standard Deviation = 158.9 seconds), producing a significant result with a t-value of 3.30 (p = 0.011). The positive value indicates that these participants were *faster* on their second attempt which was without the Reuse Assistant benefits.

Group B Analysis. Group B participants experienced the benefits of the Reuse Assistant in their second attempt and tried to solve the task without them in their first attempt. We calculated the time difference (Completion time with Reuse Assistant - Completion time without Reuse Assistant) for each participant. The mean improvement was -192.44 seconds (SD = 90.91), yielding a t-value of -6.35 (p < 0.001). This statistically significant result demonstrates that these participants were a lot *faster* on their second attempt which was with the Reuse Assistant benefits. The low standard deviation reveals that there was a consistent pattern in how much faster participants worked with the feature.

Order Effect Analysis. 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 = 6.02, p < 0.001).

The difference between the two groups was massive with a gap of 367 seconds. Group B participants, who started the task without the Reuse Assistant, finished 192 seconds faster once they had the Reuse Assistant benefits. In contrast, Group A participants started with the Reuse Assistant, but they actually took 175 seconds longer to finish compared to their subsequent performance in the unassisted condition.

This big difference points to a strong learning effect. The results show that participants were faster mainly because they got used to the problem, not just because of the benefits of the Reuse Assistant. It did not matter if they started with the Reuse Assistant or without it. The experience from the first try made the second attempt to solve the task much easier.

4.1.2 Summary of Findings. The Reuse Assistant effectively helped the end users complete the task faster. We can conclude this by comparing the average time differences and standard deviations between the two groups of participants.

Participants in Group B improved their time by an average of 192.4 seconds when they switched to using the feature. This is a larger change than Group A. Participants in Group A were slower by an average of 174.9 seconds when they stopped using the feature. This shows that the gain from adding the assistant was greater than the loss from removing it.

The standard deviation also tells us about consistency. Group B had a low standard deviation of 90.9 seconds. In contrast, Group A had a much higher standard deviation of 158.9 seconds. This means that the performance boost was not only larger but also more consistent when users utilized the Reuse Assistant.

Finally, the statistical strength confirms this result. The t-value for Group B is -6.35. This is much stronger than the t-value for Group A which is 3.30. Since the Group B result is statistically more significant, we can say that the Reuse Assistant provides a clear and robust performance advantage.

4.2 Research Question 2: To what extent does the reuse assistant facilitate reusability?

Adoption of Reusable Blocks. In the *Enhanced* OpenRoberta version that includes the Reuse Assistant feature, 18/18 participants successfully implemented a custom reusable block to handle the repetitive object placement steps. In contrast, in the *Standard* OpenRoberta version that doesn't have the Reuse Assistant benefits, participants predominantly relied on linear, repetitive code structures. Without the guidance features, none of them recognized the opportunity to create a reusable block.

4.2.1 Summary of Findings. The Reuse Assistant promotes reusability by automating the manual effort required to build reusable blocks. It achieves this by lowering the cognitive and technical barriers associated with identifying and abstracting repetitive patterns.

Specifically, the system promotes reusability through a three-step mechanism: detection, intervention, and automation. First, the feature identifies and highlights duplicate block sequences within the user's block based program. Second, it interrupts the linear workflow with a binary decision prompt, offering the user an immediate choice to refactor. Finally, upon confirmation, the system automatically encapsulates the repetitive block sequence logic into a reusable block, defines the necessary parameters if needed, and replaces the repetitive sequences with the reusable block.

The contrast in results, 0% adopted a reusable block in the Standard OpenRoberta version that does not include the Reuse Assistant versus 100% in the Enhanced version that includes the feature's benefits, demonstrates that users do not lack the ability to understand functions, but rather the initiative to implement them manually. By removing the effort required to define, parameterize, and call a function block, the Reuse Assistant transforms reusability from a complex manual task into a seamless, automated decision.

4.3 Research Question 3: How do the end-users assess the reuse assistant in terms of usability?

To answer this research question regarding the perceived usability of the system, we administered the System Usability Scale (SUS) questionnaire (see Appendix A for the full questionnaire details) to all $N = 18$ 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.

4.3.1 Overall Usability Scores. The analysis of the survey data yielded a mean SUS score of **84.0** ($Median = 81.25$). According to the interpretive ranges defined by Bangor et al. [2], a score above 80.3 is considered "Excellent" and places the system in the top 10% of products in terms of usability.

As illustrated in the figure 7, the individual scores ranged from a low of 52.5 to a perfect score of 100. Notably, 94% of participants (17 out of 18) rated the system above the industry average of 68, with the majority falling into the Excellent" or Very Good" categories.

4.3.2 Distribution Analysis. The results show that users found the system very easy to use. 17 out of 18 participants (94%) rated it above the industry average of 68. The scores mostly split into two high-performing groups: eight people rated it as "Excellent" (85–100), and nine people rated it as "Very Good" (75–80). This suggests that almost everyone understood how to use the system right away.

The scores ranged from 52.5 to 100, with a high average score of 84.0. It is worth noting that 17 of the users had scores very close to each other (between 75 and 100), which proves the system is consistently easy to use for most people.

Fig. 7. System Usability Scale (SUS) Analysis Results. (a) Distribution of SUS Scores among Participants. (b) Dual classification showing Bangor et al.'s adjective ratings (left) and Sauro & Lewis letter grades (right).

There was only one exception: a single user scored the system at 52.5. This person mentioned needing more technical help and time to learn based on this user's answers. While this shows there might be a small learning curve for some, it was an isolated case and does not change the overall positive feedback.

4.3.3 Summary of Findings. End-users assess the Reuse Assistant as an exceptionally usable system. They rate it as "Excellent" and evaluate it with a high degree of consistency.

First, users judge the system to be far superior to the industry standard. They gave it an average score of 84.0 which is much higher than the typical average of 68. This shows that users perceive the feature as significantly easier to use than standard software.

Second, users assess the system with strong agreement. 17 out of 18 participants rated the usability above 75. This tight clustering of scores proves that the positive experience was uniform across the group. Almost every user agreed that the feature was intuitive.

Third, users evaluate the system as easy to learn. The high scores indicate that participants felt confident operating the feature immediately. They did not struggle to understand how it worked. Therefore, users assess the Reuse Assistant as a polished and highly accessible solution.

4.4 Research Question 4: How do the end-users assess the perceived workload when operating the Reuse Assistant?

To assess the cognitive demands imposed by the Reuse Assistant, we administered the NASA Task Load Index (NASA-TLX) questionnaire (see Appendix B for the full questionnaire details) to all participants after completing the task with the Reuse Assistant benefits. The NASA-TLX is a widely used multidimensional assessment questionnaire that measures perceived workload across six subscales, each rated on a scale from 1 (Very Low) to 10 (Very High). The scores were calculated by multiplying each rating by 10 to convert them to a 0-100 scale.

4.4.1 Overall Workload Assessment. As illustrated in figure 8, the Reuse Assistant imposes a remarkably low workload on users, with an overall mean score of **20.00**. This indicates that the feature is highly usable and demands very little effort from the user in terms of physical or mental cost.

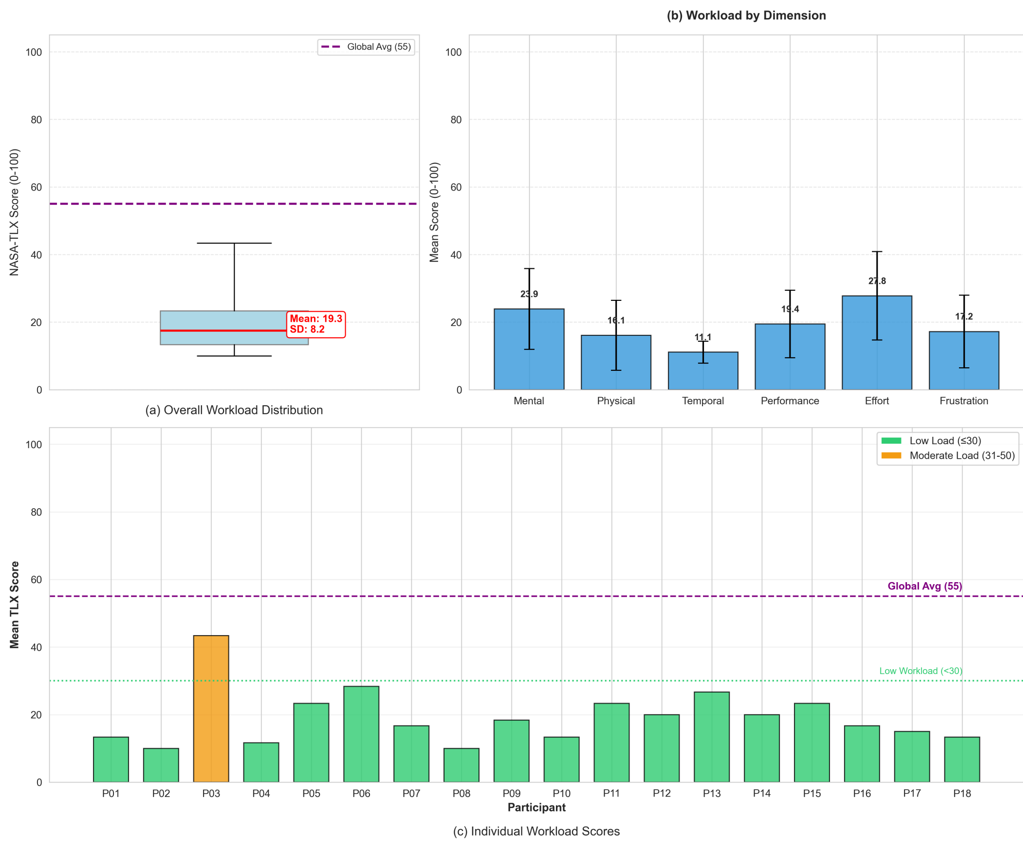

Fig. 8. NASA-tlx workload analysis results. (a) Distribution of NASA-TLX overall workload score (b) Mean workload scores for each dimension. (c) Distribution of workload scores across participants.

4.4.2 *Dimension-Specific Analysis.* There is a distinct contrast between the highest and lowest contributing factors:

- **Lowest Demand (Temporal):** With a mean score of **10.6**, Temporal Demand was negligible. This confirms that users felt **zero time pressure**, allowing them to work at their own pace without stress.
- **Highest Demand (Effort):** Effort received the highest rating (**30.00**), yet this is still considered "Low" on a 100-point scale. This suggests a positive trade-off: users were cognitively engaged (concentrating) but not overworked. The feature requires attention, but not exhaustion.

4.4.3 *Consistency and Outlier Analysis.* The data shows high consistency among the 18 participants, with one notable exception:

- **The Outlier (P03):** Participant P03 reported a "Moderate" workload (Mean: **43.3**), rating several dimensions at 5/10. This deviation suggests that while the feature is intuitive for the vast majority, a small subset of users may lack specific prerequisite knowledge or experience an edge-case interaction.
- **Consistency:** When excluding P03, the mean workload for the remaining 17 participants drops to **18.6**. Furthermore, **94%** of users (17/18) reported an overall workload below 30.0. This strong consistency statistically validates the Reuse Assistant as a low-load feature for the target population.

4.4.4 *Summary of findings.* Participants assess the perceived workload of the Reuse Assistant as remarkably low. The data indicates that operating this specific feature imposes very little mental or physical cost on the user.

Specifically, participants evaluate the feature's workload with an overall mean score of 20.0 on a scale of 100. This is exceptionally low and proves that the feature itself is easy to understand and handle.

Furthermore, participants assess the "Effort" required to use the feature as manageable. While "Effort" was the highest rated factor (30.0), it remains in the low range. This distinction is important: it shows that users were concentrating on the feature's feedback (engaged), but the interaction did not overwork them. Finally, this assessment is highly consistent, with 94% of users reporting a low workload. Therefore, participants perceive the Reuse Assistant as a supportive feature that aids their work without adding new cognitive burdens.

5 Discussion

This study evaluated the Reuse Assistant, an automated guidance feature 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 feature'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 [4] 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, as well as the risk of there being no payoff at all [5]. 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. In this study the result was 0% adoption of reusable components while using the standard OpenRoberta Lab, 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 [4] 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 [5].

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 [10] 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 [10]. 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 difficult 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 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: 1.92) and high SUS scores (mean: 84.03) 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 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 [10], 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: 84.03) place the Reuse Assistant in the "Excellent" category, with 94% of participants (17 out of 18) 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 features should prioritize clarity over complexity.

The NASA-TLX workload results (mean: 2) further support this finding, showing that effective guidance does not require complex interactions. The combination of high SUS scores and low NASA-TLX workload scores demonstrates that the Reuse Assistant successfully reduces barriers without adding cognitive burden. The key to this success is directly showing users duplicate code patterns through visual highlighting and providing one-click refactoring, rather than adding complexity.

The bimodal distribution in both SUS scores and NASA-TLX workload (with one outlier for each) 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 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. The 0% adoption rate in the standard condition demonstrates the limitations of this approach for the end-users, as participants created functional but non-optimal solutions using linear, repetitive code structures. The 100% adoption rate with the enhanced OpenRoberta version proves that tool designers must shift from providing capabilities to actively guiding their use.

5.3 Threats to Validity

5.3.1 Internal Validity.

Carryover effect. While the within subjects design allowed within-subjects comparison, the significant carryover effect ($p<0.001$) 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 417 seconds performance gap between the two groups' average time differences suggests that learning from the first condition substantially influenced performance in the second condition.

Mitigation: We explicitly analyzed and reported the carryover effect as a finding rather than treating it as unwanted noise. Furthermore, the experiments were balanced, meaning half of the participants performed the task with the Reuse Assistant first, then again using the standard OpenRoberta. The other half did the reverse sequence. In this way, the overall effect on the results is minimized.

5.3.2 External Validity.

Convenience Sampling and Population Representation. Participants were recruited through the researchers' professional networks, 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 the domain of chemistry.
- **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 18 participants for performance evaluation, usability assessment and workload assessment, the study lacks statistical power to detect small effects or to adequately characterize rare user profiles(outliers), 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, long-term 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 questionnaire, NASA-TLX questionnaire) which have established validity in usability and cognitive workload research. However:

- **SUS limitation:** SUS assesses subjective usability perception rather than objective usability metrics, such as error rates or task success beyond completion time.
- **NASA-TLX limitation:** NASA-TLX assesses subjective workload perception, which may not correlate perfectly with objective cognitive load or learning outcomes.

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 created reusable blocks when help from the Reuse Assistant was available. The feature received high usability ratings (SUS mean: 84.03) and low workload scores (NASA-TLX mean: 1.92), 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 of reusable blocks 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 test the Reuse Assistant in real laboratory settings over extended periods to determine whether users learn to recognize reuse opportunities independently. Studies with more participants from diverse backgrounds would help identify which user groups benefit most from automated guidance. The feature should also be evaluated with more complex programming tasks and tested in other end-user programming environments beyond OpenRoberta.

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] Aaron Bangor, Philip Kortum, and James T Miller. 2009. Determining what individual SUS scores mean: Adding an adjective rating scale. *Journal of usability studies* 4, 3 (2009), 114–123.

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

[4] 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

[5] 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.

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

[7] John Brooke. 1996. SUS: A 'Quick and Dirty' Usability Scale. In *Usability Evaluation in Industry*, Patrick W. Jordan, Bruce Thomas, Bernard A. Weerdmeester, and Ian L. McClelland (Eds.). Taylor & Francis, London, 189–194.

[8] Jay L Devore. 2015. *Probability and Statistics for Engineering and the Sciences* (9th ed.). Cengage Learning, Boston, MA.

[9] Sandra G Hart and Lowell E Staveland. 1988. Development of NASA-TLX (Task Load Index): Results of empirical and theoretical research. In *Human Mental Workload*, Peter A Hancock and Najmedin Meshkati (Eds.). North-Holland, Amsterdam, 139–183.

[10] 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

[11] 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

[12] 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

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

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

[15] Bernard L Welch. 1947. The generalization of 'Student's' problem when several different population variances are involved. *Biometrika* 34, 1-2 (1947), 28–35. doi:10.1093/biomet/34.1-2.28

[16] Roel J. Wieringa. 2014. *Design Science Methodology for Information Systems and Software Engineering*. Springer, Berlin, Heidelberg. doi:10.1007/978-3-662-43839-8

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

We calculate the System Usability Scale (SUS) score based on the standard method defined by Brooke [7]. The process consists of three steps:

(1) **Normalize the responses:**
   - For **odd-numbered** items (1, 3, 5, 7, 9), subtract 1 from the user response.
   - For **even-numbered** items (2, 4, 6, 8, 10), subtract the user response from 5.

   This converts every answer to a range of 0 to 4.

(2) **Sum the contributions:** Add the normalized scores for all 10 items together.

(3) **Scale the total:** Multiply the sum by 2.5 to convert the range from 0–40 to 0–100.

Formally, if x_i is the score for the i-th question, the total SUS score is given by:

$$\text{SUS Score} = 2.5 \times \left(\sum_{i \in \text{odd}} (x_i - 1) + \sum_{i \in \text{even}} (5 - x_i) \right)$$

B NASA-TLX post-task Questionnaire

The NASA Task Load Index (NASA-TLX) [9] is a multi-dimensional questionnaire that provides an overall workload score based on a weighted average of ratings on six subscales. For this study, we utilized the "Raw TLX" method (unweighted). Participants rated the following six dimensions on a scale from 1 (Very Low) to 10 (Very High), which were then normalized to a 0–100 scale.

(1) **Mental Demand:** How much mental and perceptual activity was required (e.g., thinking, deciding, calculating, remembering, looking, searching, etc.)? Was the task easy or demanding, simple or complex, exacting or forgiving?

(2) **Physical Demand:** How much physical activity was required (e.g., pushing, pulling, turning, controlling, activating, etc.)? Was the task easy or demanding, slow or brisk, slack or strenuous, restful or laborious?

(3) **Temporal Demand:** How much time pressure did you feel due to the rate or pace at which the tasks or task elements occurred? Was the pace slow and leisurely or rapid and frantic?

(4) **Performance:** How successful do you think you were in accomplishing the goals of the task set by the experimenter (or yourself)? How satisfied were you with your performance in accomplishing these goals?

(5) **Effort:** How hard did you have to work (mentally and physically) to accomplish your level of performance?

(6) **Frustration Level:** How insecure, discouraged, irritated, stressed and annoyed versus secure, gratified, content, relaxed and complacent did you feel during the task?