

1 Title 3

2 ANNE-MARIE ROMMERDAHL, SDU, Denmark

3 JEREMY ALEXANDER RAMÍREZ GALEOTTI, SDU, Denmark

4 DIMITRIOS DAFNIS, SDU, Denmark

5 NASIFA AKTER, SDU, Denmark

6 MOHAMMAD HOSEIN KARDOUNI, SDU, Denmark

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

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

9 ACM Reference Format:

10 Anne-Marie Rommerdahl, Jeremy Alexander Ramírez Galeotti, Dimitrios Dafnis, Nasifa Akter, and Mohammad Hosein Kardouni. 2018.

11 Title 3. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation email (Conference acronym 'XX).*

12 ACM, New York, NY, USA, 19 pages. <https://doi.org/XXXXXXX.XXXXXXX>

13 1 Introduction

14 2 Background and Related Work

15 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).

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

17 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.

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

19 © 2018 Copyright held by the owner/author(s). Publication rights licensed to ACM.

20 Manuscript submitted to ACM

21 Manuscript submitted to ACM

on supporting the transition to text-based languages rather than exploring how features within BBP environments [5]—such as abstraction or reuse—can enhance learning outcomes.

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[6]. 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[7], 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[8] 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 Manuscript submitted to ACM

105 search-based refactoring approach to improve the readability of Scratch programs by automatically applying small code
106 transformations, such as simplifying control structures and splitting long scripts. Their findings demonstrated that
107 automated refactoring can significantly enhance code quality and readability for novice programmers. Building upon
108 this concept, our project applies similar principles in the OpenRoberta environment, focusing on detecting duplicate
109 code segments and guiding users toward creating reusable custom blocks to promote modularity and abstraction.[1].

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

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

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

123 3 Study Design

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

127 3.1 Problem Investigation

128 *3.1.1 Problem Context and Motivation.* End-user development (EUD) for collaborative robots (cobots) presents unique
129 challenges, particularly for users without formal programming training. In domains such as chemistry laboratories,
130 educational robotics, and industrial settings, end-users need to program robots to perform specific tasks but often lack
131 the software engineering knowledge to write maintainable, well-structured code. In the domain of Chemistry, one of
132 the most relevant and important tasks is performing experiments in labs in order to test a hypothesis, or to aid in the
133 understanding of how chemicals react. Robots can be used in chemistry labs to automate experiments with great effect,
134 as many experiments involve steps that are repetitive, and susceptible to human error, such as a step being overlooked,
135 instructions being misread, etc. Automation of menial tasks will leave the chemists with more time for other work,
136 and also comes with the added bonus of chemists not having to handle dangerous chemicals. One critical challenge in
137 EUD is code reuse. Users frequently create repetitive code because they struggle to recognize duplicate patterns, lack
138 knowledge about abstraction mechanisms, or find existing tools too complex to use effectively. This problem manifests
139 in several ways: programs become unnecessarily long and difficult to maintain and small changes require modifications
140 in multiple locations, increasing the risk of errors. Several visual programming environments, like OpenRoberta Lab,
141 don't provide assistance in identifying when code should be reused or how to extract repeated sequences into reusable
142 components. As lab work in chemistry involves many repetitive tasks, these challenges can easily become an obstacle
143 for the chemists, which may turn them away from using cobots, as the inconvenience outweighs the benefits.

157 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
 158 programming experts.
 159
 160
 161
 162
 163

164 3.2 Treatment Design 165

166 To address the problem of code reuse in EUD for cobots, we have derived a set of requirements designed to contribute
 167 to the chemist's goal of creating maintainable and reusable robot programs. Functionally, the artifact must be capable
 168 of automatically detecting duplicate or similar block sequences and visually highlighting these duplications within
 169 the user's workspace. These requirements are necessary to help the end-user recognize opportunities for reuse, that
 170 would otherwise go unnoticed. Once detected, the system must suggest the creation of reusable custom blocks, allowing
 171 the user to accept or reject these suggestions. These signals are important, as they give the end-user control over the
 172 reuse process, allowing them to decide when and how to apply reuse in their programs. Regarding non-functional
 173 requirements, the artifact must seamlessly integrate with the existing Open Roberta Lab environment to ensure a
 174 smooth user experience. The interface should be intuitive for end-users, minimizing the learning curve and making it
 175 easy to understand and use the reuse features. Additionally, the artifact should not interfere with the existing workflow,
 176 allowing users to continue their programming tasks without disruption. Finally, clear visual feedback during the
 177 detection process is essential to help users understand what the system is doing and how to respond to its suggestions.
 178
 179
 180

181 3.2.1 *Artifact Specification: The Reuse Assistant.* To satisfy the requirements above, we designed the Reuse Assistant as
 182 an extension of Open Roberta Lab.
 183

184 3.2.2 *Architecture.* The system enables the execution of block-based programs on a simulated cobot through a three-tier
 185 architecture, as illustrated in 1. The workflow consists of the following stages:
 186

- 187 (1) **Client Side (Open Roberta):** The user interacts with the Open Roberta UI to assemble block sequences. The
 188 Reuse Assistant operates at this layer, analyzing blocks in real-time. Upon execution, the client generates specific
 189 data structures ("Generated Headers") representing the program logic.
 190
- 191 (2) **Backend (Flask Server):** The client transmits these headers via HTTP POST requests to a Flask-based API
 192 Endpoint. A "Translator" component processes the data, mapping the abstract block definitions to concrete
 193 Python methods compatible with the robot's control logic.
 194
- 195 (3) **Simulation (Mujoco):** The mapped methods trigger the execution of commands within the Mujoco Simulator,
 196 which renders the physical behavior of the cobot in the virtual environment.
 197

198 3.2.3 *Detection Algorithm.* The approach is intentionally simple so it is easy to read and to implement in a real block
 199 editor. The algorithm follows three main steps:
 200

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

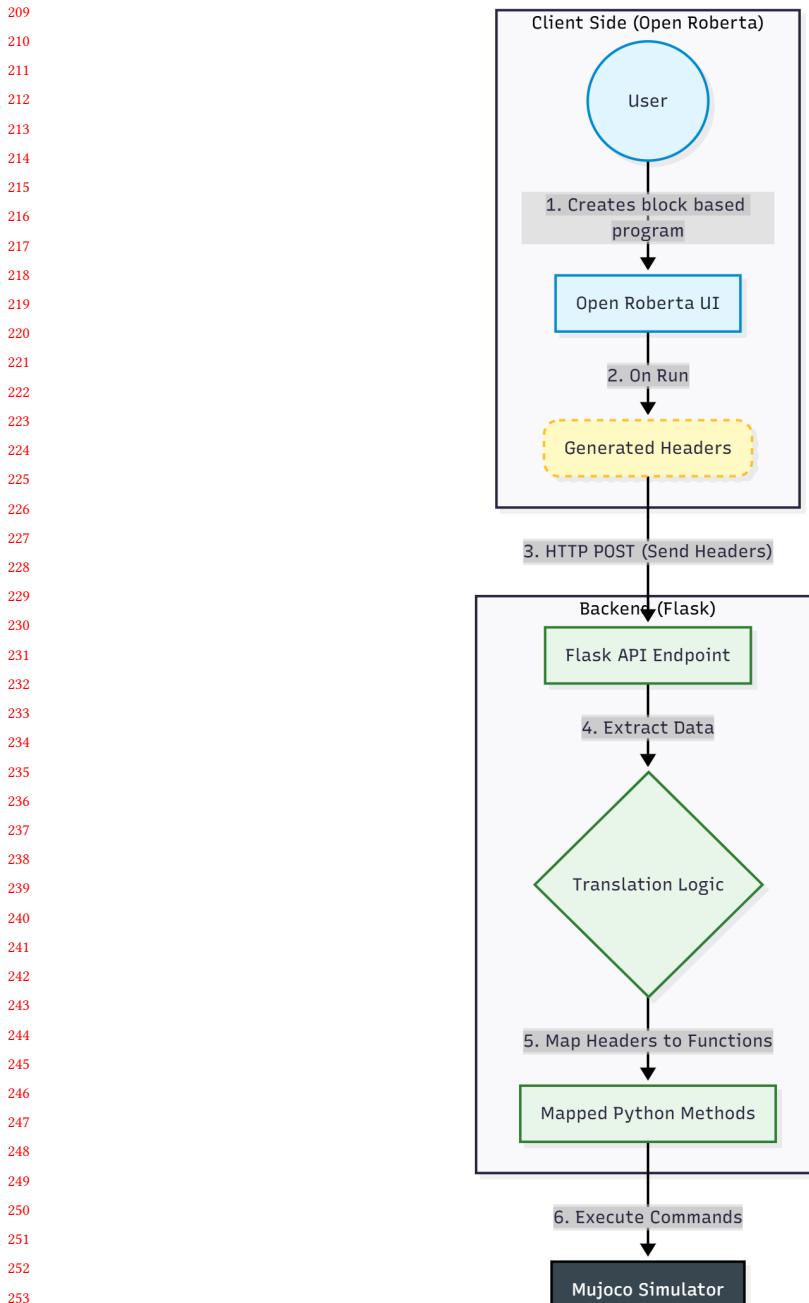


Fig. 1. System architecture

261 The pseudocode below is short, explicit, and uses straightforward data structures (lists).
 262

Algorithm 1 Duplicate Sequence Detection

263 **Require:** Workspace, StartBlock // user's block workspace
 264
 265 **Require:** MinimumSequenceLength = 3, MinimumDifferentBlockTypesInSequence = 3, MaxSequenceLength = 10
 266
 267 **Ensure:** ReusableComponentCandidates // list of repeated block sequences to return
 268
 269 1: Chain = **buildLinearChain**(StartBlock)
 270
 271 2: Sequences = List<sequence>
 272 3: **for** startIndex = 0 **to** length(Chain) - 1 **do**
 273 4: **for** sequenceLength = 1 **to** MaxSequenceLength **do**
 274 5: sequence = Chain[startIndex .. startIndex + sequenceLength - 1]
 275 6: numberofBlockTypesInSequence = getNumberOfDistinctBlockTypes(sequence)
 276 7: **if** sequenceLength >= MinimumSequenceLength **and** numberofBlockTypesInSequence >= MinimumDifferentBlockTypesInSequence **then**
 277 8: Sequences.append(sequence) // record sequence occurrence
 278 9: **end if**
 280
 281 10: **end for**
 282
 283 11: **end for**
 284
 285 12: ReusableComponentCandidates = {Sequences | occurrence \geq 2}
 286 13: sort ReusableComponentCandidates by (longest sequence length and most recent occurrence)
 287 14: **return** ReusableComponentCandidates[0] // Return highest priority candidate

288
 289 Algorithm 1. Illustrates the core logic for identifying duplicate block sequences

290
 291 3.2.4 *User Interface and Interaction.* The user interface is designed to be intuitive and non-disruptive. When the
 292 detection algorithm identifies a candidate, the system visually highlights the blocks on the canvas as illustrated in
 293 Figure 2. A non-blocking toast notification appears, prompting the user to confirm the refactoring. If confirmed, the
 294 system automatically generates the custom block definition in a dedicated workspace area (handling visibility via
 295 revealDefinitionWorkspacePane) and updates the main workspace, replacing the redundant code with concise
 296 function calls as shown in Figure 3. This process abstracts the complexity of manual function creation, guiding the user
 297 toward modular design practices. After the user presses the run simulation button, the robot simulator of mujoco opens
 298 up and executes the commands provided by the user inside the Open Roberta workspace. This is illustrated in Figure 4.
 299
 300

301
 302
 303
 304
 305
 306
 307
 308
 309
 310
 311
 312

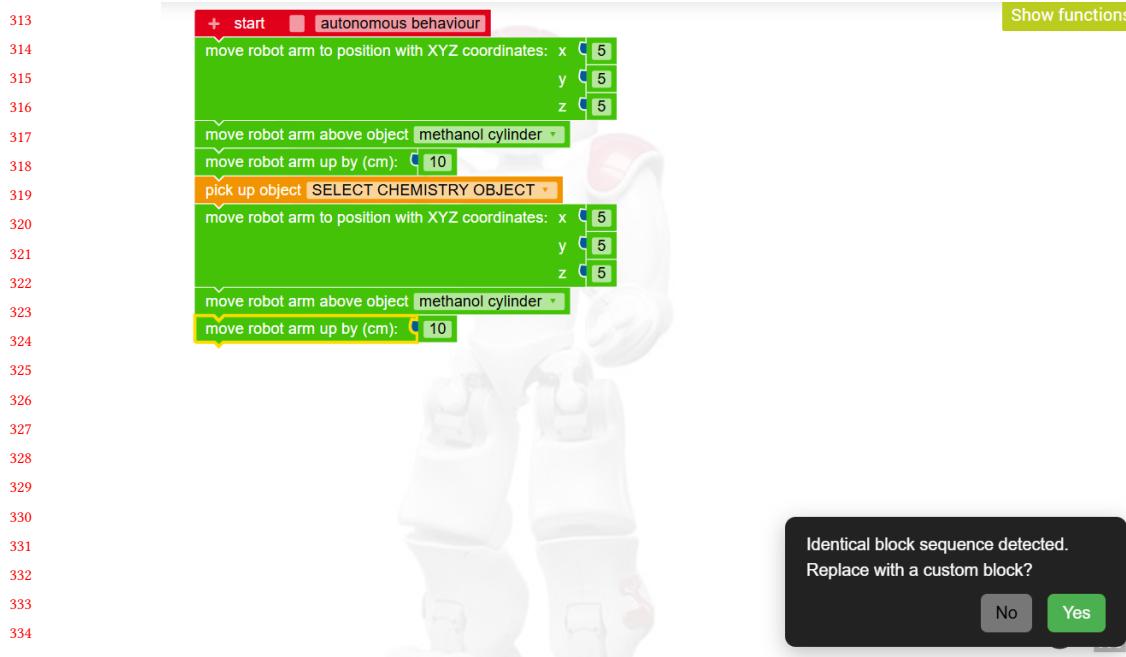


Fig. 2. Reuse Assistant workflow — detection: the interface detects and highlights duplicate blocks by changing their color to green.

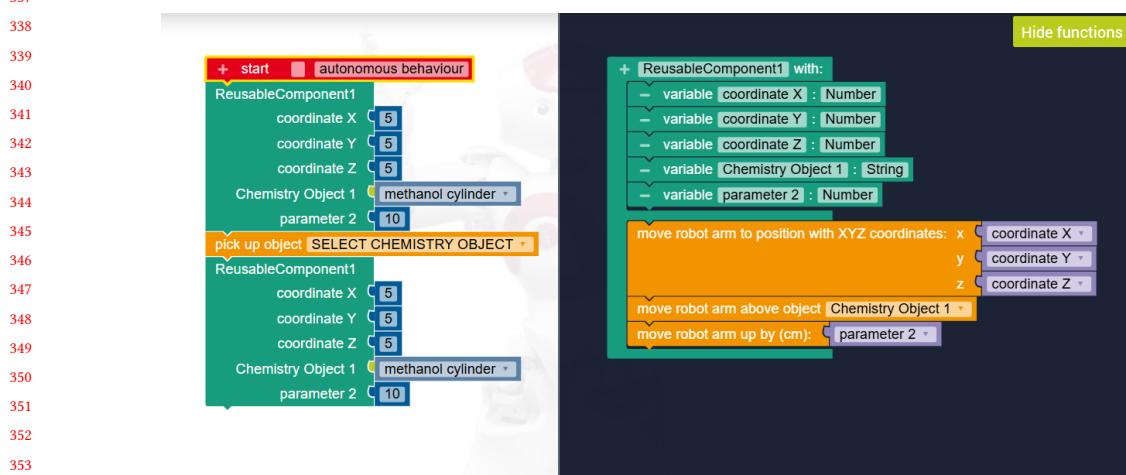


Fig. 3. Reuse Assistant workflow — refactoring: the automated refactoring result, showing the new custom block definition and the simplified main program.

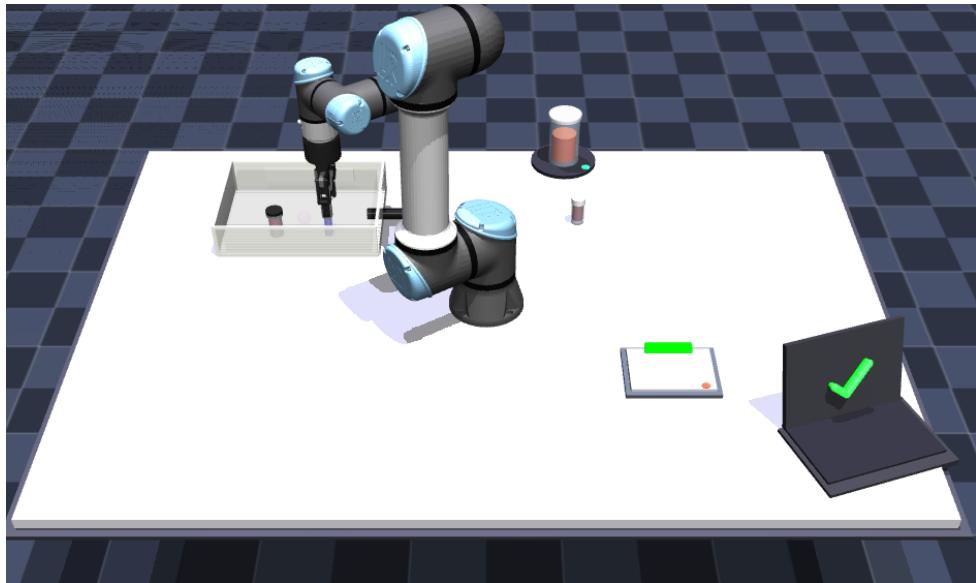


Fig. 4. Mujoco robot simulator executing the commands from Open Roberta.

365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416

417 3.3 Treatment Validation

418
419 The treatment validation for this study adopts a mixed-methods evaluation approach to assess the effectiveness of
420 the proposed features for guiding users in creating custom reusable components (blocks) within the OpenRoberta
421 environment.

422
423 3.3.1 *Participant Recruitment.* A total of 10 participants will be selected with similar level of expertise in block-based
424 programming. Time constraints and resource availability have influenced the decision to limit the number of participants.
425 Participants will be recruited from a diverse pool of individuals affiliated with the University of Southern Denmark
426 and the broader chemistry community. This group of participants includes chemistry teachers, professional chemical
427 engineers, and students currently enrolled in chemistry-intensive curricula. To ensure relevant practical expertise, the
428 selection specifically targets those who frequently engage in laboratory environments. The experimental sessions will
429 be conducted across a range of environments to accommodate participant availability. Physical sessions will take place
430 within the chemistry laboratories at the University of Southern Denmark (SDU) as well as a private residential setting.
431 For remote participants, sessions will be administered virtually using Discord for communication and AnyDesk for
432 remote desktop control.
433

434
435
436 3.3.1 *Ethical Considerations and Sampling.* Prior to the commencement of the study, all participants are required to sign a
437 consent form acknowledging their voluntary participation and granting permission for screen recording and data usage.
438 It should be noted that this recruitment strategy constitutes *convenience sampling*. As such, they may not represent the
439 general population.
440

441
442 3.3.2 *Task Execution.* The participants will initially be given a short introduction to the OpenRoberta UI, as well
443 as the mujoco robot simulator. They will then perform one task which is described by a set of pre-defined steps to
444 perform. This task has been specifically designed to promote the reusability aspect. The task is focused on the domain
445 of chemistry, as it is modelled after a real lab experiment performed by chemistry students at SDU.
446

447 The participants will be instructed to program the robot to execute the following sequence of operations:
448

- 449 (1) Move the robot arm above mix cylinder
- 450 (2) Mix the chemistry ingredients
- 451 (3) Move the robot arm above the analysis pad
- 452 (4) Analyze the sample
- 453 (5) If the solution is analyzed (use if statement) then show a response message in the laptop's screen
- 454 (6) Place the following three objects into their corresponding slots in the chemistry equipment toolbox:
 - 455 • Methanol cylinder
 - 456 • Chloroform syringe
 - 457 • Toluene syringe
- 458 (7) Important notes for the participants:
 - 459 • After placing an object to its slot in the toolbox **wait 2 seconds** before you move to pick a new one.
 - 460 • 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
 - 461 • Click the **play** button on the bottom right corner to start the simulation
 - 462 • Click the **reset** button on the bottom right corner to reset the scene of the robot simulator

469 Most optimal solution pre-defined by the researchers:

470

471

472

473

474

475

476

477

478

479

480

481

482

483

484

485

486

487

488

489

490

491

492

493

494

495

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

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

504
 505
 506 3.3.3 *Data Gathering and Analysis.* Data collection focuses on both quantitative performance and qualitative feedback
 507 from participants:

- 508
 509 (1) **Task Completion Time:** Comparing the participants who will first use the enhanced version of OpenRoberta
 510 against those who will first use the standard version.
- 511 (2) **Solution Accuracy:** Evaluated by comparing the participant's block configuration against the pre-defined
 512 optimal solution.
- 513 (3) **Survey Feedback:** Collected via post-experiment surveys designed to capture demographic data and subjective
 514 perceptions of the effectiveness and usability of the block creation guidance feature.

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

517
 518 Manuscript submitted to ACM

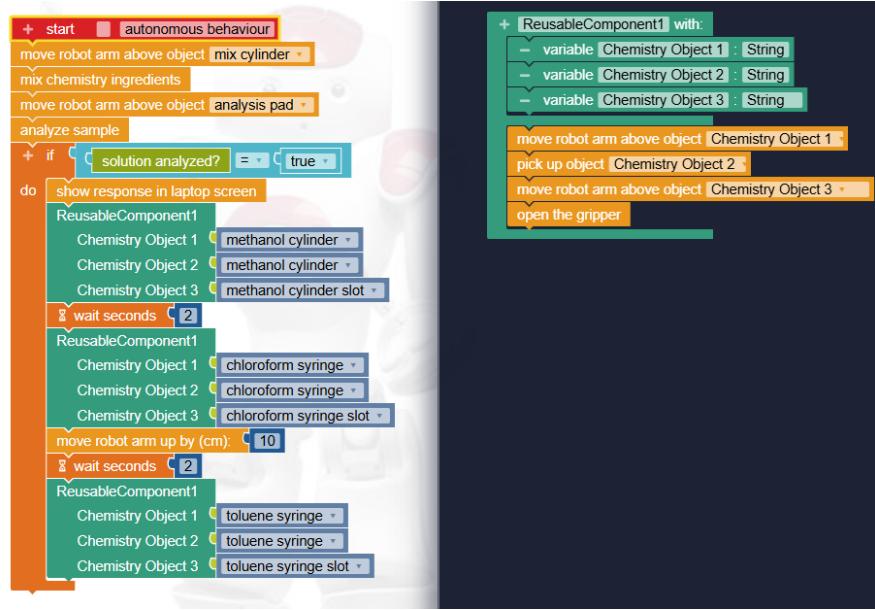


Fig. 5. The optimal solution implemented in OpenRoberta, utilizing a custom block for the object placement sequence.

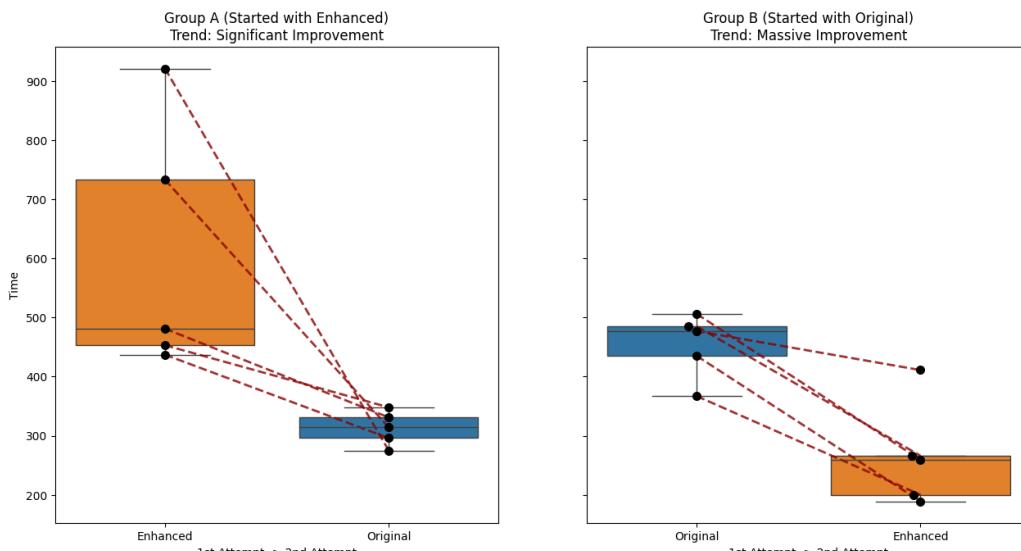
521 **4 Results**

522 **4.1 Research Question 1: Can the Reuse Assistant improve the end-users performance?**

523

525 Participant ID	526 Group (Order)	527 Completion time (Enhanced)	528 Completion time (Original)	529 Time Difference
P01	A (Enhanced → Original)	481 seconds	331 seconds	150 seconds
P03	A (Enhanced → Original)	921 seconds	275 seconds	646 seconds
P06	A (Enhanced → Original)	733 seconds	314 seconds	419 seconds
P07	A (Enhanced → Original)	437 seconds	296 seconds	141 seconds
P09	A (Enhanced → Original)	453 seconds	348 seconds	105 seconds
P02	B (Original → Enhanced)	411 seconds	477 seconds	-66 seconds
P04	B (Original → Enhanced)	189 seconds	435 seconds	-246 seconds
P05	B (Original → Enhanced)	200 seconds	367 seconds	-167 seconds
P08	B (Original → Enhanced)	266 seconds	485 seconds	-219 seconds
P10	B (Original → Enhanced)	259 seconds	506 seconds	-247 seconds

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



531 Fig. 6. Boxplot of task completion times

532 **4.1.1 Statistical Analysis.** To evaluate the effectiveness of the Enhanced version, we conducted paired t-tests on the
533 completion times. The analysis reveals distinct patterns between groups and identifies a significant order effect.

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

Overall Comparison. When combining all 10 participants regardless of order, the overall mean difference was -51.60 seconds (StandardDeviation = 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 ($SD = 234.19$), 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 ($t = -4.37, df \approx 5, p = .008$).

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.

4.2 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 $N = 11$ participants immediately following the treatment validation.

625 The SUS yields a single number representing a composite measure of the overall usability of the system, with scores
 626 ranging from 0 to 100.
 627

628 4.2.1 *Overall Usability Scores.* The analysis of the survey data yielded a mean SUS score of **84.1** (*Median* = 80.0).
 629 According to the interpretive ranges defined by Bangor et al., a score above 80.3 is considered “Excellent” and places
 630 the system in the top 10% of products in terms of usability.
 631

632 As detailed in Table 4, the individual scores ranged from a low of 52.5 to a perfect score of 100. Notably, 90% of
 633 participants (9 out of 10) rated the system above the industry average of 68, with the majority falling into the “Excellent”
 634 or “Very Good” categories.
 635

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

651 Table 4. Individual System Usability Scale (SUS) Scores
 652
 653
 654

655 4.2.2 *Distribution Analysis.* The SUS scores demonstrate a strong positive skew, with 9 out of 10 participants (91%)
 656 rating the system above the industry average of 68. The distribution reveals tight clustering in two distinct bands: five
 657 participants (45%) achieved scores in the “Excellent” range (92.5-100), while another five participants (45%) scored in
 658 the “Very Good” range (75-80). This bimodal clustering pattern, with no scores between 52.5 and 75, suggests that users
 659 either readily adopted the system’s paradigm or encountered initial conceptual barriers.
 660

661 The score range spans 47.5 points (52.5 to 100), with the median (80.0) slightly below the mean (84.1), indicating
 662 that high-scoring participants pull the average upward. The consistency among the top 10 participants is particularly
 663 noteworthy, with all scores falling within a 25-point band, demonstrating reliable usability for the vast majority of the
 664 target user population. The single outlier at 52.5, while representing only 9% of participants, warrants investigation as
 665 it may identify a specific user profile or interaction pattern requiring additional support.
 666

667 One outlier was observed (Participant 3, Score: 52.5), who indicated needing technical support and additional learning
 668 time. This suggests a slight learning curve for users less comfortable with block-based encapsulation concepts, though
 669 as an isolated case, it does not negate the general consensus of high usability.
 670

672 4.3 Research Question 3: What is the perceived workload when using the Reuse Assistant?

673 To assess the cognitive demands imposed by the Reuse Assistant, we administered the NASA Task Load Index (NASA-
 674 TLX) questionnaire to all participants after completing the task with the enhanced version. The NASA-TLX is a widely
 675

677 used multidimensional assessment tool that measures perceived workload across six subscales, each rated on a scale
 678 from 1 (Very Low) to 10 (Very High).
 679

680 **4.3.1 Overall Workload Assessment.** The analysis of NASA-TLX data yielded an overall mean workload score of **1.88**
 681 across all six dimensions. According to NASA-TLX interpretation guidelines, scores below 2.0 indicate very low perceived
 682 workload, suggesting that the Reuse Assistant imposed minimal cognitive and physical burden on users.
 683

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

699 Table 5. NASA-TLX Workload Scores by Participant and Dimension
 700
 701
 702

703 **4.3.2 Dimension-Specific Analysis.** As shown in Table 5, the six NASA-TLX dimensions assessed were:
 704

- 705 • **Mental Demand:** How much mental and perceptual activity was required to understand and use the Reuse
 706 Assistant? (Mean: 2.2)
- 707 • **Physical Demand:** How much physical activity was required (e.g., clicking, modifying program) while using
 708 the Reuse Assistant? (Mean: 1.9)
- 709 • **Temporal Demand:** How much time pressure did you feel while completing the task using the Reuse Assistant?
 710 (Mean: 1.2)
- 711 • **Performance:** How successful do you think you were in accomplishing the goals using the Reuse Assistant?
 712 (Mean: 1.9)
- 713 • **Effort:** How hard did you have to work to accomplish your level of performance when using the Reuse Assistant?
 714 (Mean: 2.8)
- 715 • **Frustration:** How insecure, discouraged, irritated, stressed and annoyed were you when using the Reuse
 716 Assistant feature? (Mean: 1.7)

720 The temporal demand dimension received the lowest mean rating (1.2), indicating that participants experienced
 721 minimal time pressure. The effort dimension received the highest rating (2.8), though still well within the low workload
 722 range, suggesting that while some concentration was required, it remained manageable for most users.
 723

724 **4.3.3 Outlier Consideration.** Consistent with the SUS findings, Participant P03 reported significantly elevated workload
 725 across most dimensions (mean: 4.33), with ratings of 5 (Very High) for mental demand, physical demand, effort, and
 726 frustration. This participant corresponds to the same individual who reported the lowest SUS score (52.5), reinforcing
 727
 728 Manuscript submitted to ACM

729 the pattern that a small subset of users may require additional support or training to effectively utilize the system's
730 automation features.

731 Excluding this outlier, the remaining nine participants showed remarkably consistent low workload scores (mean:
732 1.58), with eight of nine reporting individual mean scores below 2.0. This consistency demonstrates that the Reuse
733 Assistant successfully minimizes cognitive burden for the vast majority of the target user population.
734

735 5 Discussion

736 This study evaluated the Reuse Assistant, an automated guidance tool designed to help end-users recognize and
737 implement code reuse in block-based programming environments. Through a crossover study with 10 participants from
738 the chemistry domain, we assessed the tool's impact on performance (RQ1), usability (RQ2), and perceived workload
739 (RQ3). The findings reveal both the potential and limitations of automated assistance in promoting software reuse
740 practices among domain experts with limited programming expertise.
741

742 5.1 Implications for Theory

743 5.1.1 *Addressing the Recognition Barrier in End-User Development.* The results provide empirical evidence for a critical
744 distinction between two types of barriers to software reuse: the *recognition barrier* and the *selection barrier*. The
745 recognition barrier refers to users' inability to identify opportunities for reuse, while the selection barrier concerns
746 difficulties in choosing or implementing appropriate reuse mechanisms once an opportunity is recognized.
747

748 Our findings demonstrate that the recognition barrier is the primary obstacle for our end-users. In the standard
749 OpenRoberta version, despite the task being deliberately designed with repetitive sequences, zero participants (0%)
750 created reusable components. This was not due to lack of capability, the same participants achieved 100% adoption
751 when using the Enhanced version with automated detection. This stark contrast suggests that domain experts possess
752 the cognitive capacity to understand and apply reuse concepts, but lack the pattern recognition skills that professional
753 developers develop through experience.
754

755 This finding extends Ko et al.'s [4] learning barriers framework by demonstrating that in block-based environments,
756 the recognition barrier precedes and dominates over other barriers. The low NASA-TLX workload scores (mean: 1.88)
757 and high SUS scores (mean: 84.1) indicate that once reuse opportunities are identified for users, the selection and
758 implementation processes impose minimal cognitive burden.
759

760 5.1.2 *The Order Effect: Prior Experience as a Prerequisite for Appreciating Automation.* The significant order effect
761 ($t=-4.37$, $p=.008$) reveals a counter-intuitive finding: participants who received automated assistance first were actually
762 slower to complete tasks than those who first struggled with the manual approach. This 481-second performance gap
763 suggests that automation effectiveness depends on users having established mental models of the problem space.
764

765 This finding has theoretical implications for understanding how end-users learn to value productivity tools. Participants
766 in Group B (Original → Enhanced) developed an experiential baseline that allowed them to recognize what the
767 automation was helping them avoid. In contrast, Group A participants (Enhanced → Original) lacked this reference
768 frame, potentially viewing the automated suggestions as interruptions rather than assistance.
769

770 This aligns with theories of learning transfer and expertise development, suggesting that some exposure to manual
771 processes may be pedagogically valuable before introducing automation. It challenges the assumption that "easier is
772 always better" in tool design, indicating that cognitive struggle during initial learning may enhance appreciation and
773 effective utilization of advanced features.
774

781 5.1.3 Low Workload Despite High Effectiveness. The NASA-TLX results (mean: 1.88, with temporal demand at 1.2
782 and frustration at 1.7) demonstrate that effective guidance does not require complex interactions. This challenges
783 assumptions that powerful automation must impose cognitive overhead. The Reuse Assistant achieves its impact
*784 through simple visual highlighting and one-click acceptance, suggesting that the key to reducing barriers is *making the**
*785 *invisible visible* rather than increasing system sophistication.*

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

793 5.2 Implications for Practice

794 5.2.1 Design Principle 1: From Passive Toolboxes to Active Assistants. Current block-based programming environments
795 (Scratch, Blockly, standard OpenRoberta) follow a passive interaction model where reuse mechanisms exist as features
796 waiting to be discovered. Our 0% adoption rate in the standard condition demonstrates the failure of this approach
797 for end-user developers. The 100% adoption rate with automated detection proves that tool designers must shift from
798 providing capabilities to actively guiding their use.

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

804 5.2.2 Design Principle 2: Strategic Introduction of Automation. The order effect findings have direct implications for
805 training and onboarding. Organizations introducing automated coding assistants should consider implementing a
806 staged approach:

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

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

814 5.2.3 Design Principle 3: Minimize Interaction Complexity. The exceptionally low NASA-TLX scores (temporal demand:
815 1.2, frustration: 1.7) demonstrate that effective guidance need not be complex. The Reuse Assistant succeeded through:

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

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

837
838 5.2.4 *Design Principle 4: Plan for Individual Differences.* The consistent outlier pattern (one participant with low
839 SUS scores and high workload across all dimensions) indicates that approximately 10% of users may struggle with
840 automated guidance. This is likely unavoidable given individual differences in learning preferences and comfort with
841 system-initiated interactions.

842
843 **Practical Recommendation:** Provide a clearly visible mechanism to disable automated suggestions for users who
844 find them distracting or confusing. Additionally, supplement automated detection with alternative pathways (manual
845 invocation, documentation, tutorials) to ensure users who reject proactive guidance can still access reuse mechanisms
846 if they choose to seek them out.

847 5.3 Threats to Validity

848 5.3.1 Internal Validity.

849
850 *Order Effects and Carryover.* While the crossover design allowed within-subjects comparison, the significant order
851 effect ($p=.008$) indicates that the sequence of conditions fundamentally altered the user experience. This carryover
852 effect means we cannot cleanly separate the impact of the Reuse Assistant from the impact of prior experience. The
853 481-second performance gap between groups suggests that learning from the first condition substantially influenced
854 performance in the second condition.

855
856 *Mitigation:* We explicitly analyzed and reported the order effect as a finding rather than treating it as unwanted
857 noise. The crossover design, despite this limitation, provided valuable insights about how prior experience shapes users'
858 ability to benefit from automation. Future studies could employ between-subjects designs to isolate tool effects, though
859 this would sacrifice statistical power given small sample sizes typical in EUD research.

860
861 *Task Specificity.* The experimental task, while modeled on authentic chemistry lab procedures, represented a single
862 workflow pattern with clear repetitive sequences. The Reuse Assistant's effectiveness may not generalize to tasks with:

- 863 • Less obvious repetition patterns
- 864 • Longer or more complex reusable sequences
- 865 • Multiple potential refactoring opportunities requiring prioritization
- 866 • Nested or hierarchical code structures

867
868 *Mitigation:* The task was designed in consultation with chemistry educators and reflected realistic laboratory
869 automation scenarios. However, broader validation across diverse task types and domains is necessary to establish
870 generalizability.

871 5.3.2 External Validity.

872
873 *Convenience Sampling and Population Representation.* Participants were recruited through the researchers' professional
874 networks at the University of Southern Denmark, constituting a convenience sample. This introduces several limitations:

- 875 • **Geographic concentration:** All participants affiliated with a single institution in Denmark, potentially
876 reflecting local educational approaches and cultural attitudes toward automation.

- **Self-selection bias:** Volunteers may be more technologically inclined or motivated than typical chemistry professionals who would use cobot programming in practice.
- **Limited domain diversity:** While participants included chemistry teachers, students, and professional chemical engineers, they do not represent the full spectrum of potential end-user developers in laboratory automation.
- **Sample size:** With N=10 for performance evaluation and N=11 for usability assessment, the study lacks statistical power to detect small effects or to adequately characterize rare user profiles (such as the consistent outlier).

Implications: Findings should be interpreted as preliminary evidence rather than definitive 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.

Future work: Complement these measures with code quality metrics (complexity, maintainability indices), comprehension assessments (asking users to explain or modify reusable components), and longitudinal evaluation of whether users internalize reuse concepts or remain dependent on automated detection.

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

- 937 • Fatigue or external stressors unrelated to the tool
 938 • Genuine fundamental incompatibility between the user's mental model and the tool's interaction paradigm
 939

940 *Limitation:* With only one outlier, we cannot determine which explanation is correct or whether this represents 10%
 941 of the population or a unique case. Larger samples are needed to characterize the distribution of user experiences.
 942

943 References

- 944 [1] Felix Adler, Gordon Fraser, Eva Gründinger, Nina Körber, Simon Labrenz, Jonas Lerchenberger, Stephan Lukasczyk, and Sebastian Schweikl. 2021.
 945 Improving Readability of Scratch Programs with Search-Based Refactoring. In *Proceedings of the IEEE/ACM 43rd International Conference on Software*
 946 *Engineering: Companion Proceedings (ICSE-SEET)*. IEEE. doi:10.1109/ICSE-Companion.2021.00105
 947 [2] Len Bass, Paul Clements, and Rick Kazman. 2021. *Software Architecture in Practice, 4th Edition*. Addison-Wesley Professional.
 948 [3] Alexander Bock and Ulrich Frank. 2021. Low-Code Platform. *Business and Information Systems Engineering* 63 (2021). doi:10.1007/s12599-021-00726-8
 949 [4] Andrew J. Ko, Brad A. Myers, and Htet Htet Aung. 2004. Six Learning Barriers in End-User Programming Systems. *IEEE Symposium on Visual*
 950 *Languages and Human-Centric Computing (VL/HCC)* (2004), 199–206. doi:10.1109/VLHCC.2004.47
 951 [5] Yuhua Lin and David Weintrop. 2021. The Landscape of Block-Based Programming: Characteristics of Block-Based Environments and How They
 952 Support the Transition to Text-Based Programming. *Journal of Computer Languages* 67 (2021), 101075. doi:10.1016/j.jola.2021.101075
 953 [6] Hugo Lourenço, Carla Ferreira, and João Costa Seco. 2021. OSTRICH - A Type-Safe Template Language for Low-Code Development. In *2021 ACM/IEEE*
 954 *24th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. 216–226. doi:10.1109/MODELS50736.2021.00030
 955 [7] Vlad Magdalin. 2012. Low code platform tool Webflow. <https://webflow.com/>.
 956 [8] Derek Roos. 2005. Low code platform tool Mendix. <https://www.mendix.com/>.
 957

958 A System Usability Scale (SUS) Questionnaire

959 The System Usability Scale (SUS) is a widely used standardized questionnaire for assessing the perceived usability of
 960 a system. Participants respond to each statement using a 5-point Likert scale ranging from 1 (Strongly Disagree) to
 961 5 (Strongly Agree). The SUS score is calculated by converting the responses to a scale of 0-100, where higher scores
 962 indicate better usability.
 963

964 A.1 SUS Statements

- 965 (1) I think that I would like to use the Reuse Assistant feature frequently.
 966 (2) I found the Reuse Assistant feature unnecessarily complex.
 967 (3) I thought the Reuse Assistant feature was easy to use.
 968 (4) I think that I would need the support of a technical person to be able to use the Reuse Assistant feature.
 969 (5) I found the various functions in the Reuse Assistant feature were well integrated.
 970 (6) I thought there was too much inconsistency in the Reuse Assistant feature.
 971 (7) I would imagine that most people would learn to use the Reuse Assistant feature very quickly.
 972 (8) I found the Reuse Assistant feature very cumbersome to use.
 973 (9) I felt very confident using the Reuse Assistant feature.
 974 (10) I needed to learn a lot of things before I could get going with the Reuse Assistant feature.
 975

976 A.2 Scoring Method

977 For odd-numbered items (1, 3, 5, 7, 9), the score contribution is the scale position minus 1. For even-numbered items (2,
 978 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
 979 obtain the overall SUS score, which ranges from 0 to 100.
 980

981 Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009
 982