

1 **Guiding End-Users towards Software Reuse: An Evaluation of Automated
2 Assistance in Block-Based Programming for Chemistry Laboratory Automation**
3

4 ANNE-MARIE ROMMERDAHL, SDU, Denmark
5

6 JEREMY ALEXANDER RAMÍREZ GALEOTTI, SDU, Denmark
7

8 DIMITRIOS DAFNIS, SDU, Denmark
9

10 NASIFA AKTER, SDU, Denmark
11

12 MOHAMMAD HOSEIN KARDOUNI, SDU, Denmark
13

14 **Abstract**—Background: End-users who program collaborative robots for laboratory automation often create repetitive
15 code because they struggle to recognize opportunities for reuse. While block-based programming environments provide
16 accessible interfaces, they do not actively guide users toward creating reusable components.
17

18 Objective: This study investigates whether automated guidance can help end-users recognize and apply code reuse
19 practices. We developed the Reuse Assistant, a feature that automatically detects duplicate code sequences within the
20 OpenRoberta environment and guides users to create reusable custom blocks through visual highlighting and one-click
21 refactoring.
22

23 Study Design: Through a within subjects study with 18 participants from the chemistry domain, we evaluated the feature's
24 impact on performance, usability, and perceived workload.
25

26 Results: Automated guidance increased reuse adoption from 0% in the standard OpenRoberta version to 100% when
27 using the Reuse Assistant. The feature achieved high usability scores (SUS mean: 84.03) and imposed minimal cognitive
28 burden (NASA-TLX mean score: 1.92). The significant carryover effect revealed that prior manual experience helps users
29 appreciate automation benefits.
30

31 Conclusions: The dramatic shift in adoption proves that users are capable of using advanced features if the system
32 actively guides them. However, the order effect suggests that some manual experience is necessary to fully benefit from
33 automation. Participants who struggled with manual coding first developed a mental model that allowed them to better
34 appreciate and efficiently use the automated assistance.
35

36 **ACM Reference Format:**
37

38 Anne-Marie Rommerdahl, Jeremy Alexander Ramírez Galeotti, Dimitrios Dafnis, Nasifa Akter, and Mohammad Hosein Kardouni. 2018.
39 Guiding End-Users towards Software Reuse: An Evaluation of Automated Assistance in Block-Based Programming for Chemistry
40 Laboratory Automation. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation email (Conference*
41 *acronym 'XX)*. ACM, New York, NY, USA, 25 pages. <https://doi.org/XXXXXX.XXXXXXX>
42

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

46 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
47 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
48 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
49 servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
50

51 © 2018 Copyright held by the owner/author(s). Publication rights licensed to ACM.
52

53 Manuscript submitted to ACM
54

55 Manuscript submitted to ACM
56

53 1 Introduction

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

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

64 This study addresses the question: Can automated guidance help end-users recognize and apply code reuse in
 65 block-based programming? We developed the Reuse Assistant, a feature that automatically detects duplicate code
 66 sequences and guides users to create reusable custom blocks through visual highlighting and one-click refactoring.
 67 Through a within subjects study with 18 participants from the chemistry domain, we evaluated whether proactive
 68 automated assistance can overcome the barriers that prevent end-users from adopting reuse practices.

69 Our investigation examined three research questions: (1) Can the Reuse Assistant improve the end-users' performance?
 70 (2) Is the Reuse Assistant sufficiently usable for end-users? (3) What is the perceived workload when using the Reuse
 71 Assistant? The results showed that automated guidance increased reuse adoption from 0% to 100%, achieved high
 72 usability scores (SUS mean: 84.03), and imposed minimal cognitive burden (NASA-TLX mean score: 2).

73 The contributions of this work are both theoretical and practical. We extend the Attention Investment Model [4]
 74 and Learning Barriers Framework [7] by demonstrating that proactive assistance transforms feature adoption from a
 75 high-cost investment to a low-cost opportunistic choice, effectively eliminating the selection barrier. However, we also
 76 identify a critical "order effect," suggesting that manual experience is a prerequisite for maximizing the efficiency of
 77 automated tools. From a practical perspective, our results show that while simple design principles (visual highlighting,
 78 one-click acceptance) achieve high usability, the most effective adoption occurs when automation is introduced after
 79 users have developed a mental model of the manual process.

80 2 Background and Related Work

81 Software reuse is a broad term, that refers to the practice of reusing previously written code, rather than coding from
 82 scratch. It is such an important part of software engineering, that one of the ways to measure the quality of software
 83 is by its 'Reusability'[3], i.e. the degree to which the application or its components can be reused. There are multiple
 84 benefits to practicing reuse in software engineering. One developer could save time by using another developer's
 85 reusable component, rather than coding their own. The developer avoids both the work of writing the syntax and
 86 designing the logic of the component. The developer can design their own reusable components, keeping all the logic
 87 in one place, which can then be tested thoroughly. However, despite reuse being an important practice in software
 88 engineering, there is still a limited focus on this practice when it comes to low-code development platforms (LCDP).

89 A study from 2021 studied several low-code platforms (LCPs), in order to identify characteristic features of LCPs.
 90 The identified features were presented according to how frequent they occurred, with domain-specific reference artifacts
 91 being categorized as 'rare'. Most studied systems offered catalogs of "reusable functions or examples of predefined

105 processes", but they were found to be generic, or have a limited scope[6]. This lack of focus on promoting reuse may
 106 impact the so-called 'Citizen Developers', who have little or no coding knowledge, and whom may then miss out on the
 107 benefits of reuse. Lin and Weintrop (2021) noted that most existing research on block-based programming focuses on
 108 supporting the transition to text-based languages rather than exploring how features within BBP environments [8],
 109 such as abstraction or reuse, can enhance learning outcomes.
 110

111 There have been proposed some ideas on how to promote reuse for LCPs, such as the templating language OSTRICH,
 112 developed for model-driven low-code platform OutSystems[9]. OSTRICH is designed to assist the end-user in making
 113 use of OutSystems' available templates, by abstracting and parameterizing the templates. However, OSTRICH only
 114 supports the top nine most used production-ready screen templates, and does not allow the end-user to create and
 115 save their own templates, or re-apply a template which they have customized. Another approach focused on enabling
 116 the reuse of models, by providing recommendations to the end-user, based on the models stored in a graph acting as
 117 a repository. While the graph allows end-users to reuse their own models, there is no mention of guiding the user
 118 towards reusing their own models.
 119

120 Several popular low-code development platforms (LCDPs) provide different kinds of support for reuse. Webflow[10],
 121 a LCDP for responsive websites, offers the ability to create reusable components and UI kits, which can be reused across
 122 multiple pages and projects. Mendix[11] and OutSystems offer even more functionality to support reuse, offering several
 123 ways to end-users to share their code with each other, and offering pre-made components. Both of these platforms
 124 also utilize AI to enhance reuse. Outsystems provides AI suggestions to spot and create reusable pieces, while Mendix
 125 uses AI to suggest the best solutions and components for specific tasks. However, for both of these platforms, the AI
 126 suggestions provided are not always accurate to successfully guide the end-user to create custom reusable components.
 127 In order to analyze how block-based robotics environments address reuse, 4 representative platforms were compared:
 128 mBlock, MakeCode, SPIKE LEGO, VEXcode GO and Open Roberta. The comparison focused on three main dimensions
 129 of reuse: structural reuse (through user-defined blocks or functions), social reuse (through sharing or remixing existing
 130 projects), and interoperable reuse (through import/export capabilities).
 131

132 Table 1. Block Based Robotics Environments Reuse Support
 133

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

147 In this context, "reuse support" represents a scale that measures how effectively each platform facilitates reuse-related
 148 features. High reuse support indicates that users can easily create, share, and adapt existing components or projects.
 149 Medium reuse support suggests that some reuse mechanisms are available but limited in scope or flexibility. Low reuse
 150 support implies that the platform provides only minimal or restricted features to promote reuse.
 151

152 As shown in Table 1, although these platforms include reusability features, they are quite limited, as none of them
 153 provide users with clear guidance on how to use these tools effectively, which restricts their ability to fully leverage them.
 154

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

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

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

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

3 Study Design

Following the Design Science methodology [12], our study is structured into three main phases: problem investigation to define goals, treatment design to specify the artifact requirements, and treatment validation to assess the artifact's performance in a controlled environment.

3.1 Problem Investigation

3.1.1 Problem Context and Motivation. End-user development (EUD) for collaborative robots (cobots) presents unique challenges, particularly for users without formal programming training. In domains such as chemistry laboratories, educational robotics, and industrial settings, end-users need to program robots to perform specific tasks but often lack the software engineering knowledge to write maintainable, well-structured code. In the domain of Chemistry, one of the most relevant and important tasks is performing experiments in labs in order to test a hypothesis, or to aid in the understanding of how chemicals react. Robots can be used in chemistry labs to automate experiments with great effect, as many experiments involve steps that are repetitive, and susceptible to human error, such as a step being overlooked, instructions being misread, etc. Automation of menial tasks will leave the chemists with more time for other work, and also comes with the added bonus of chemists not having to handle dangerous chemicals.

One critical challenge in EUD is code reuse. According to Blackwell's Attention Investment Model [4], users decisions about whether to engage in programming activites, are based on an implicit cost-benefit analysis. How big an investment must be made to create the program, how great will the pay-off be, and what is the risk that the investment won't pay off. While the usage of reusable components can be a great way to lower the cost of creating a program, the decision whether to create such a component is also subject to a cost-benefit analysis. Several visual programming environments,

like OpenRoberta Lab, don't provide assistance in identifying when code should be reused or how to extract repeated sequences into reusable components. This leaves end-users with the added cost of having to figure out how each platform supports reuse, and how to make use of the provided features. 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, as the investment and risk outweighs the benefits.

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

3.2 Treatment Design

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

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

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

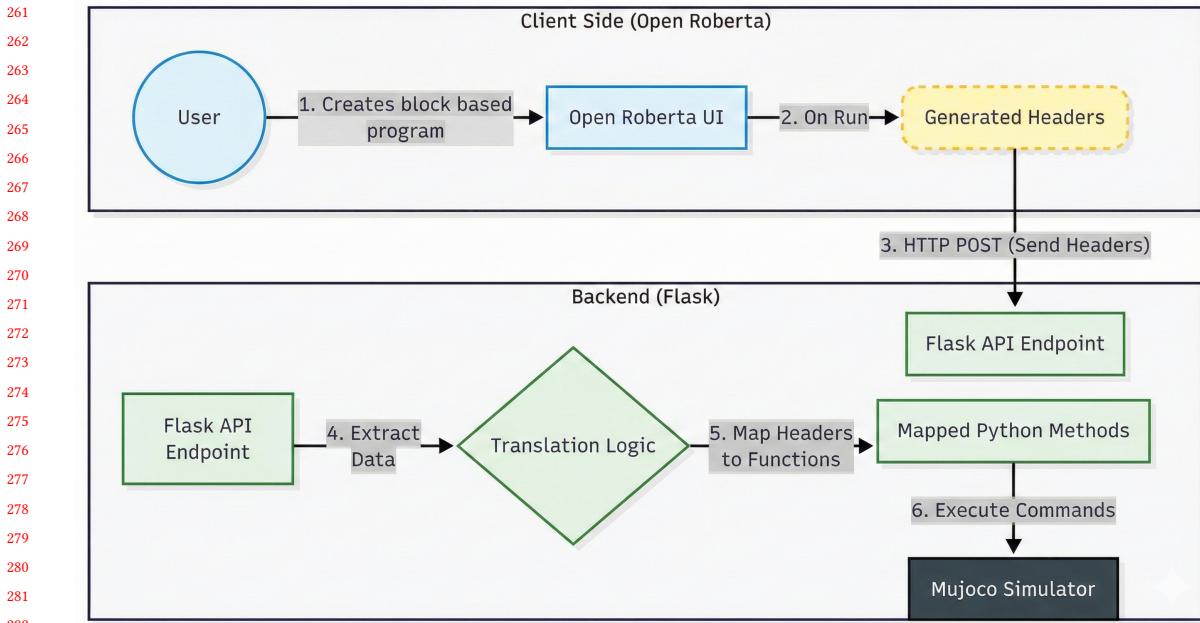


Fig. 1. System architecture: Data flow from Client Side to Simulator

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

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

310 The pseudocode below is short, explicit, and uses straightforward data structures (lists).
311
312 Manuscript submitted to ACM

313 **Algorithm 1** Duplicate Sequence Detection

314 **Require:** Workspace, StartBlock // user's block workspace

315 **Require:** MinimumSequenceLength = 3, MinimumDifferentBlockTypesInSequence = 3, MaxSequenceLength = 10

316 **Ensure:** ReusableComponentCandidates // list of repeated block sequences to return

317 1: Chain = **buildLinearChain**(StartBlock)

318 2: Sequences = List<sequence>

319 3: **for** startIndex = 0 **to** length(Chain) - 1 **do**

320 4: **for** sequenceLength = 1 **to** MaxSequenceLength **do**

321 5: sequence = Chain[startIndex .. startIndex + sequenceLength - 1]

322 6: numberOfBlockTypesInSequence = getNumberOfDistinctBlockTypes(sequence)

323 7: **if** sequenceLength >= MinimumSequenceLength **and** numberOfBlockTypesInSequence >= MinimumDifferentBlockTypesInSequence **then**

324 8: Sequences.append(sequence) // record sequence occurrence

325 9: **end if**

326 10: **end for**

327 11: **end for**

328 12: ReusableComponentCandidates = {Sequences | occurrence \geq 2}

329 13: sort ReusableComponentCandidates by (longest sequence length and most recent occurrence)

330 14: **return** ReusableComponentCandidates[0] // Return highest priority candidate

337
338 Algorithm 1. Illustrates the core logic for identifying duplicate block sequences

339
340 3.2.4 *User Interface and Interaction.* The user interface is designed to be intuitive and non-disruptive. When the
341 detection algorithm identifies a candidate, the system visually highlights the blocks on the canvas as illustrated in
342 Figure 2. A non-blocking toast notification appears, prompting the user to confirm the refactoring. If confirmed, the
343 system automatically generates the custom block definition in a dedicated workspace area (handling visibility via
344 revealDefinitionWorkspacePane) and updates the main workspace, replacing the redundant code with concise
345 function calls as shown in Figure 3. This process abstracts the complexity of manual function creation, guiding the user
346 toward modular design practices. After the user presses the run simulation button, the robot simulator of mujoco opens
347 up and executes the commands provided by the user inside the Open Roberta workspace. This is illustrated in Figure 4.
348
349
350

351
352
353
354
355
356
357
358
359
360
361
362
363
364

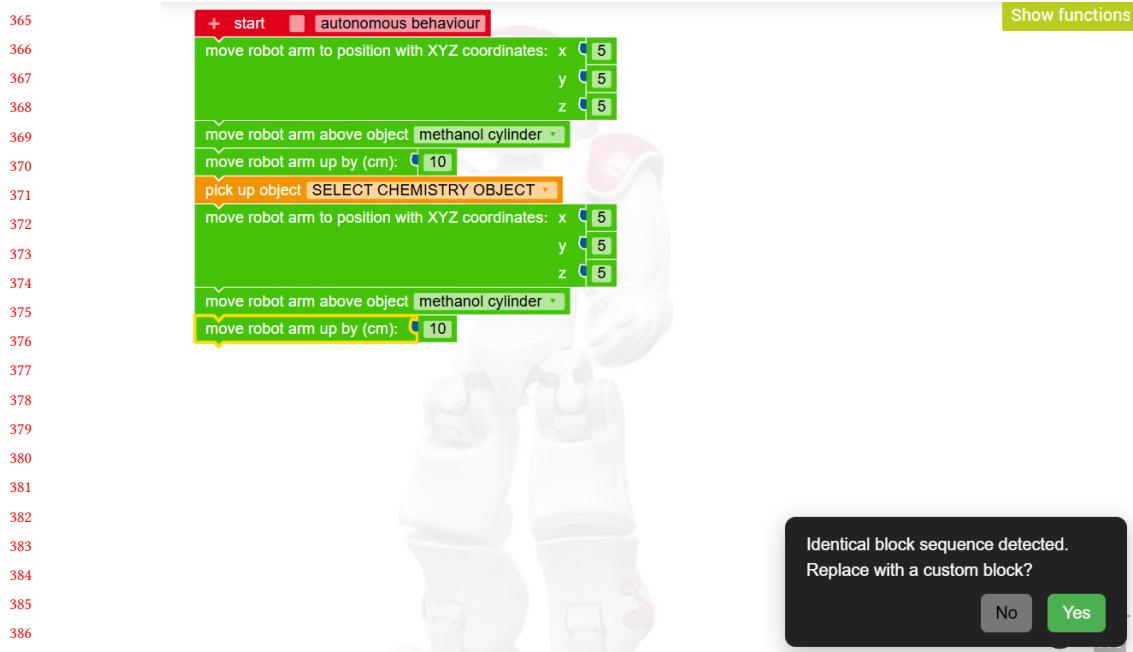


Fig. 2. Reuse Assistant workflow: detection - the interface detects and highlights duplicate blocks by changing their color to green.

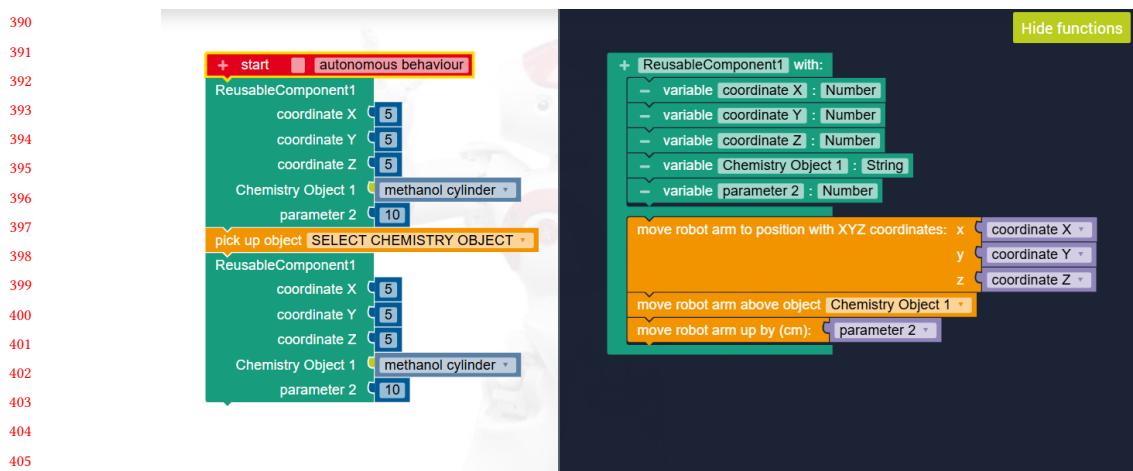


Fig. 3. Reuse Assistant workflow: refactoring - the automated refactoring result, showing the new custom block definition and the simplified main program.

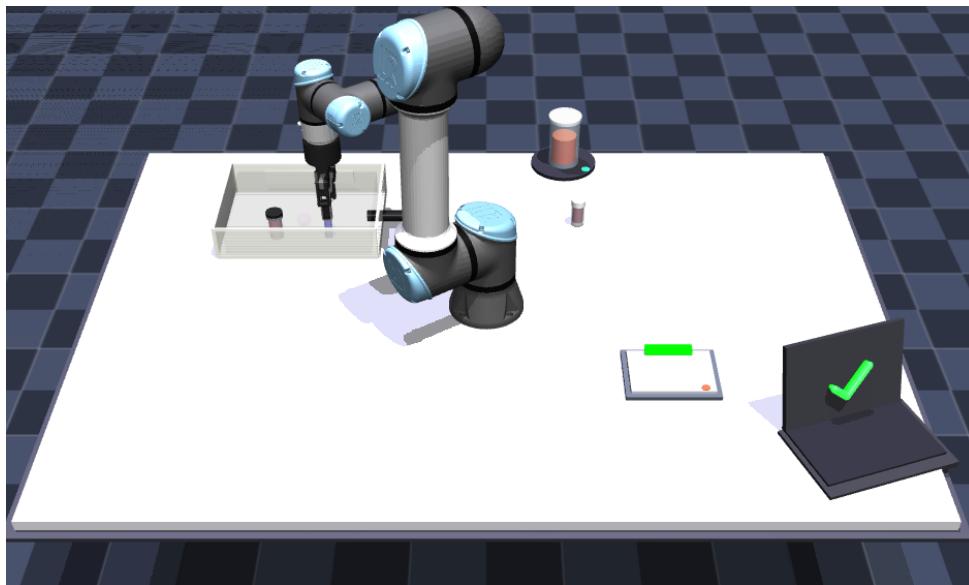


Fig. 4. Mujoco robot simulator executing the commands from Open Roberta.

469 3.3 Treatment Validation

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

473
474
475 3.3.1 *Participant Recruitment.* A total of 18 participants were selected with similar level of expertise in block-based
476 programming. Time constraints and resource availability have influenced the decision to limit the number of participants.
477 Participants were recruited from a diverse pool of individuals affiliated with the University of Southern Denmark
478 and the broader chemistry community. This group of participants includes chemistry teachers, professional chemical
479 engineers, and students currently enrolled in chemistry-intensive curricula. To ensure relevant practical expertise, the
480 selection specifically targets those who frequently engage in laboratory environments. The experimental sessions were
481 conducted across a range of environments to accommodate participant availability. Physical sessions took place within
482 the chemistry laboratories at the University of Southern Denmark (SDU) as well as a private residential setting. For
483 remote participants, sessions were administered virtually using Discord for communication and AnyDesk for remote
484 desktop control.
485
486

487
488 *Ethical Considerations and Sampling.* Prior to the commencement of the study, all participants are required to sign a
489 consent form acknowledging their voluntary participation and granting permission for screen recording and data usage.
490 It should be noted that this recruitment strategy constitutes *convenience sampling*. As such, they may not represent the
491 general population.
492
493

494 3.3.2 *Task Execution.* The participants were initially given a short introduction to the OpenRoberta UI, as well as the
495 mujoco robot simulator. They then performed one task which is described by a set of pre-defined steps to perform. This
496 task has been specifically designed to promote the reusability aspect. The task is focused on the domain of chemistry,
497 as it is modelled after a real lab experiment perfomed by chemistry students at SDU.
498
499

500 The participants were instructed to program the robot to execute the following sequence of operations:

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

521 Most optimal solution pre-defined by the researchers is illustrated in Figure 5.

522

523

524

525

526

527

528

529

530

531

532

533

534

535

536

537

538

539

540

541

542

543

544

545

546

547

548

549

550

551

552

553

554

555

556

557

558

559

560

561

562

563

564

565

566

567

568

569

570

571

572

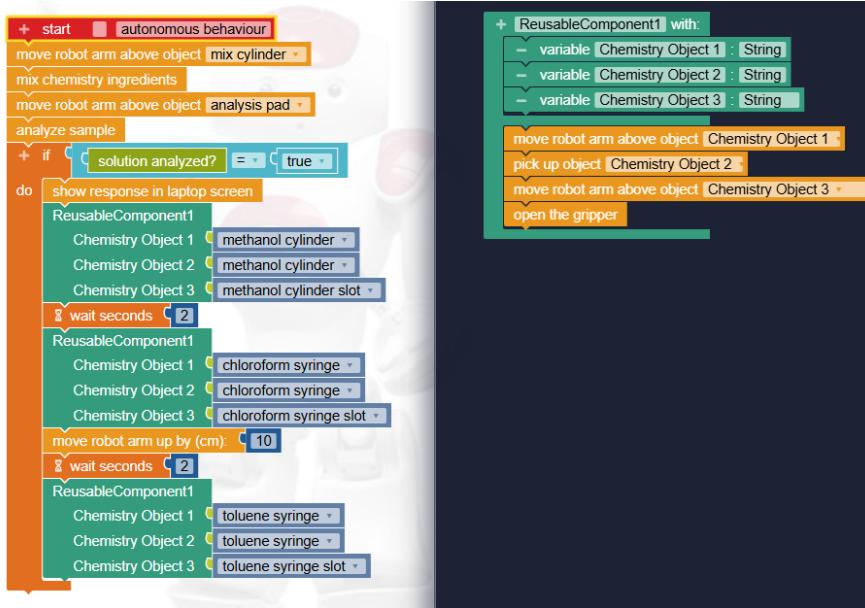


Fig. 5. The optimal solution implemented in OpenRoberta, utilizing a custom block for the object placement sequence.

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

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

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

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

- 573 (4) **Workload Assessment:** Measured using the NASA post-task Workload questionnaire (NASA-TLX) to assess
574 the cognitive demands imposed by the Reuse Assistant across six dimensions (mental demand, physical demand,
575 temporal demand, performance, effort, and frustration).
576

577 This comprehensive evaluation provided a detailed understanding of how useful and effective is the Reuse Assistant
578 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 learning transfer between the two conditions.

Table ?? presents the individual completion times for all participants across both conditions. The data reveal substantial variability in performance outcomes depending on the presentation order, with Group B participants showing consistent improvements when moving from Original to Enhanced, while Group A participants exhibited the opposite pattern.

Participant ID	Completion time (Enhanced version)	Completion time (Original version)	Time Difference
P01	481 seconds	331 seconds	150 seconds
P03	921 seconds	275 seconds	646 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	-140 seconds
P17	560 seconds	440 seconds	120 seconds

Table 2. Task Completion Times: Group A (Solved the task first with Enhanced and then moved to Original)

Participant ID	Completion time(Enhanced version)	Completion time(Original version)	Time 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: Group B (Original → Enhanced)

Figure 6 illustrates the impact of the carryover effect on performance. The data reveal that participants who used the Enhanced version had significantly reduced completion times for Group B while Group A participants experienced slow times with the enhanced version and showed improved performance with the standard.

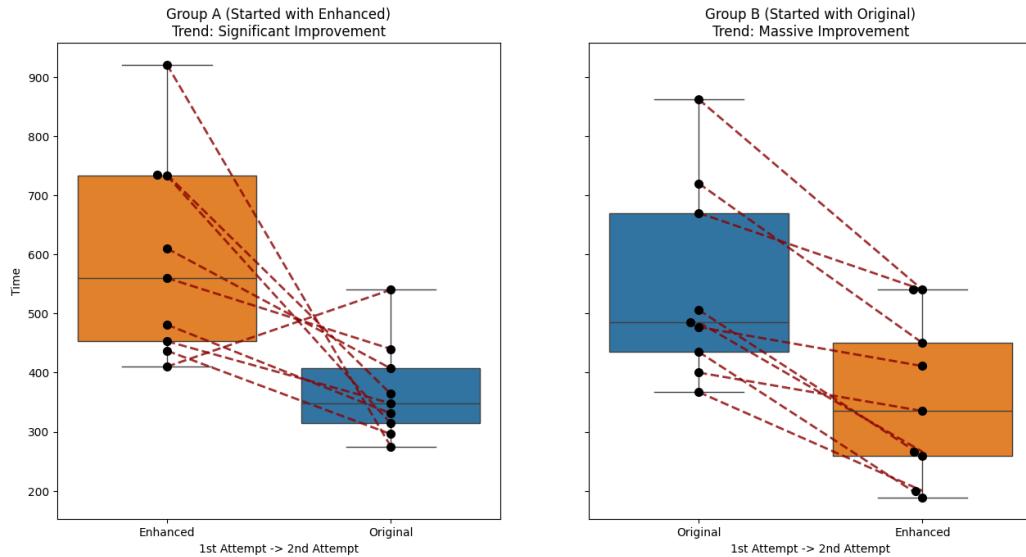


Fig. 6. Boxplot of task completion times

To statistically evaluate these observed differences, we conducted paired t-tests and a Welch's t-test 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 (carryover effect).

Test	t-Value	p-value
Overall Comparison	0.25	0.802
Group A Improvement	3.02	0.017
Group B Improvement	-6.35	< 0.001
Carryover Effect	5.18	< 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 overall mean time difference was +16.28 seconds (Standard Deviation = 271.28), leading to a t-value = 0.25 ($p = 0.805$). This non-significant result indicates no overall difference when order is ignored.

Group B Analysis (Original → Enhanced). Group B participants started with the Original version and then used the Enhanced version of OpenRoberta. We calculated the difference (Original - Enhanced) for each participant, with negative values indicating faster performance on Enhanced. 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 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 the Enhanced version and switched to Original. The mean difference (Original - Enhanced) was +225.0 seconds (Standard Deviation = 234.19), producing a t-value of 3.02 ($p = 0.017$). The positive 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 feature'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 ($t = 5.18, p < 0.001$).

The magnitude of this effect is substantial: there was a gap of 417 seconds between the two groups' mean improvement scores. Group B participants, who started with the Original version, showed an average improvement of -192.44 seconds when they switched to Enhanced. In contrast, Group A participants, who started with Enhanced, showed an average change of +225.0 seconds (meaning they were actually slower on Enhanced). This creates a total difference of approximately 417 seconds between the groups' experiences.

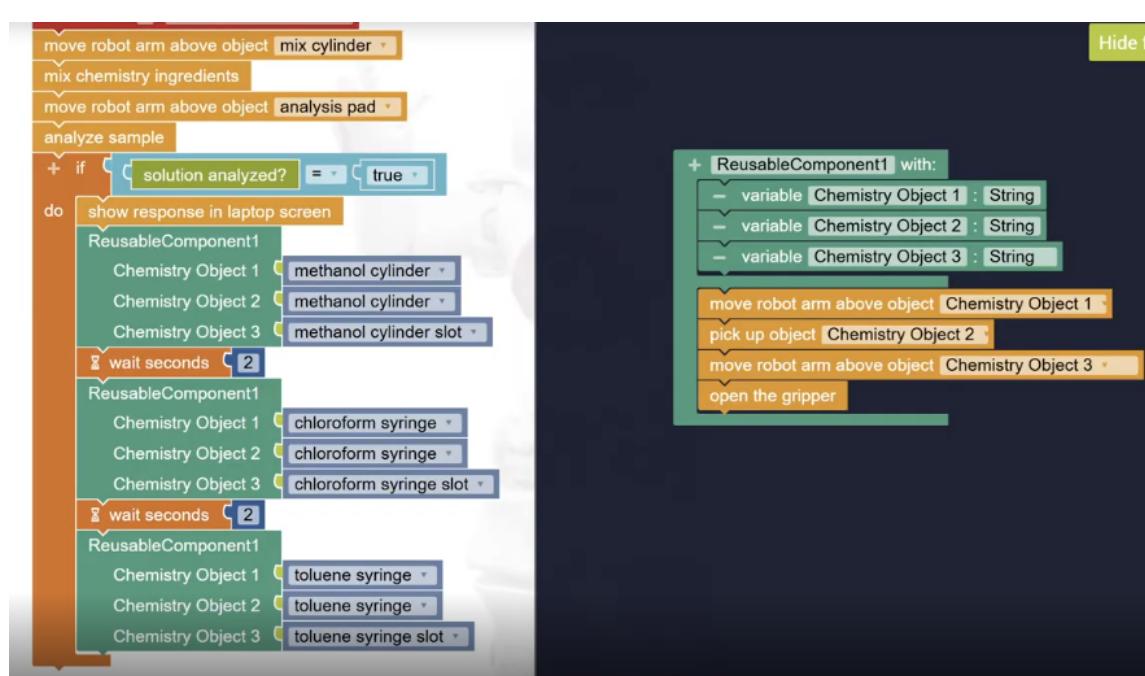
This finding demonstrates that presentation order profoundly impacts user performance. Participants who first struggled with the original version (Group B) were able to recognize and appreciate the value of the automated reuse feature when they encountered them second. Conversely, participants who received 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.

729 **4.2 Research Question 2: To what extent does the reuse assistant facilitate reusability?**

730 **4.2.1 How much does the reuse assistant promote reusability?** Solution accuracy was evaluated by comparing participant
 731 solutions against the optimal reference solution defined in the treatment validation (see Section 3.3).

732 **Adoption of Reusable Blocks.** A key metric was the adoption of the custom reusable component. In the *Enhanced*
 733 version, 18/18 participants successfully implemented a custom reusable block to handle the repetitive object placement
 734 steps. In contrast, in the *Standard* condition, participants predominantly relied on linear, repetitive code structures.
 735 Without the guidance features, none of them recognized the opportunity to create a reusable block.

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



770 Fig. 7. Example of a different participant's solution compared to the optimal solution.

771 **4.3 Research Question 3: How do the end-users assess/experience the reuse assistant in terms of usability?**

772 To answer the third research question regarding the perceived usability of the system, we administered the System
 773 Usability Scale (SUS) questionnaire to all $N = 18$ participants immediately following the treatment validation.

774 The SUS yields a single number representing a composite measure of the overall usability of the system, with scores
 775 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 detailed in Table 5, 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.

Participant ID	SUS Score	Adjective Rating
P01	100.0	Excellent
P02	100.0	Excellent
P03	97.5	Excellent
P04	95.0	Excellent
P05	95.0	Excellent
P06	92.5	Excellent
P07	87.5	Excellent
P08	85.0	Excellent
P09	82.5	Very Good
P10	80.0	Very Good
P11	80.0	Very Good
P12	80.0	Very Good
P13	80.0	Very Good
P14	77.5	Very Good
P15	77.5	Very Good
P16	75.0	Very Good
P17	75.0	Very Good
P18	52.5	OK
Mean Score	84.03	Excellent
Median Score	81.25	Very Good
Highest Score	100.0	Excellent
Lowest Score	52.5	OK

Table 5. System Usability Scale (SUS) Scores

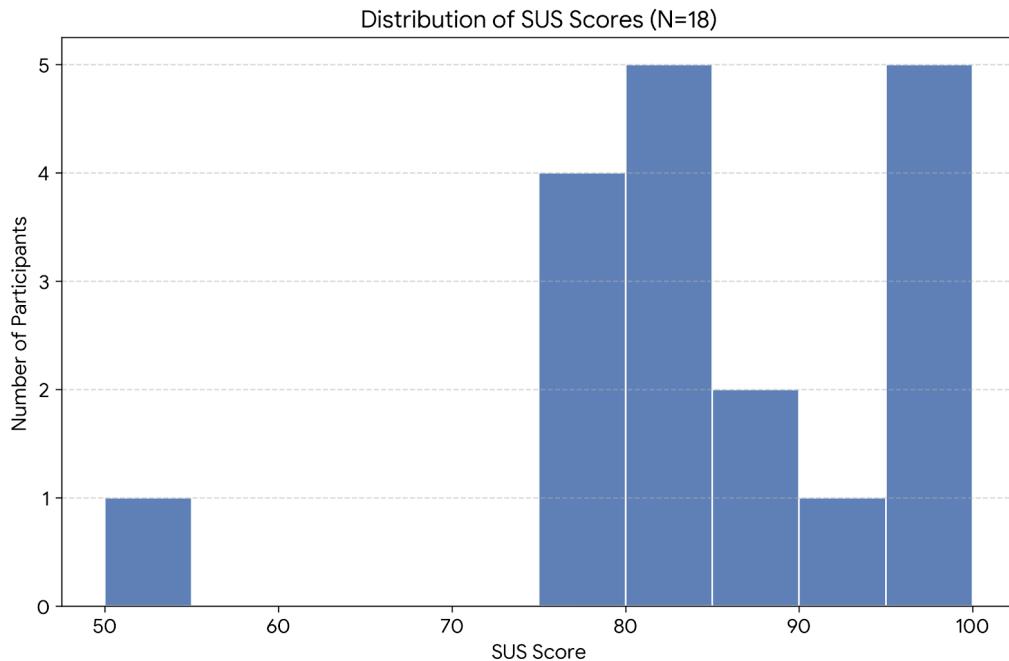


Fig. 8. Distribution of SUS Scores

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" (75–82.5). 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.

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. 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.

4.4 Research Question 4: How do participants 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 to all participants after completing the task with the enhanced version. 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).

885 *4.4.1 Overall Workload Assessment.* The Reuse Assistant imposes a remarkably low workload on users, with an overall
886 mean score of **2.00** on a scale of 10. This indicates that the feature is highly usable and demands very little effort from
887 the user in terms of physical or mental cost.
888

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
P11	3	2	1	2	4	2	2.33
P12	3	1	1	2	3	2	2.00
P13	4	1	1	2	5	3	2.67
P14	2	2	1	3	3	1	2.00
P15	3	2	1	2	3	3	2.33
P16	2	1	1	2	3	1	1.67
P17	3	1	1	2	1	1	1.50
P18	1	1	2	1	2	1	1.33
Overall Mean Score:							1.92

Table 6. Workload Scores by Participant and Dimension

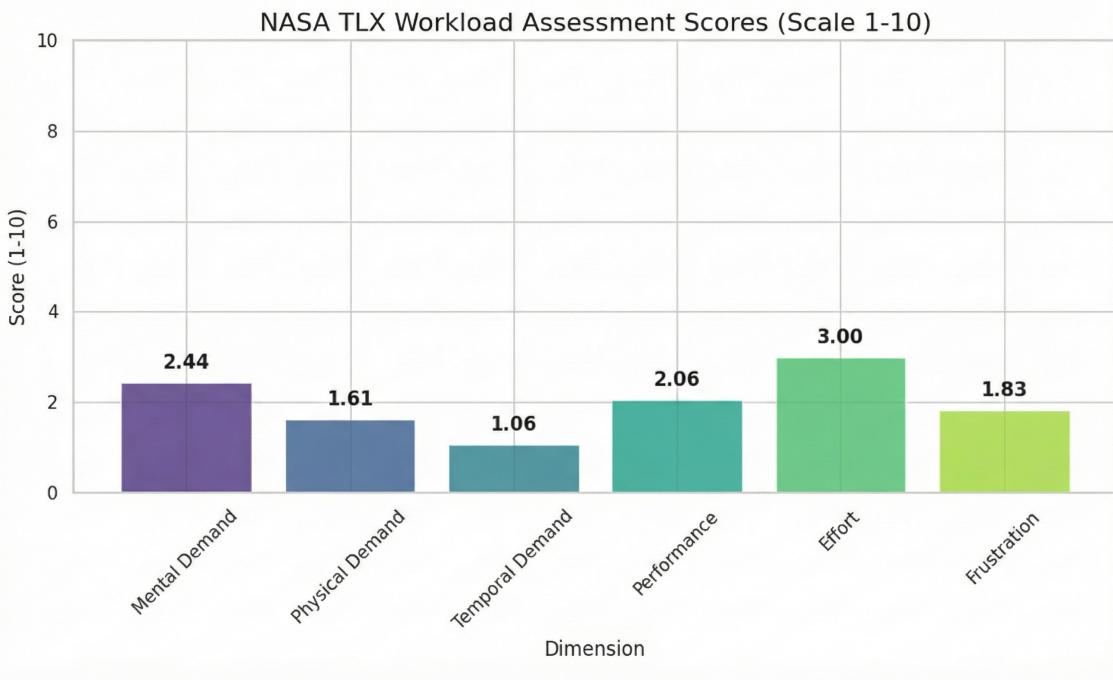


Fig. 9. Workload mean scores for the Reuse Assistant on each dimension. Scores are presented on a scale of 1-10.

4.4.2 *Dimension-Specific Analysis*. As shown in Table 6, the six NASA-TLX dimensions assessed were:

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

There is a distinct contrast between the highest and lowest contributing factors:

- **Lowest Demand (Temporal):** With a mean score of 1.06, Temporal Demand was negligible. This confirms that users felt **zero time pressure**, allowing them to work at their own pace without stress.

- 989 • **Highest Demand (Effort):** Effort received the highest rating (**3.00**), yet this is still considered “Low” on
 990 a 10-point scale. This suggests a positive trade-off: users were cognitively engaged (concentrating) but not
 991 overworked. The feature requires attention, but not exhaustion.

993 4.4.3 *Consistency and Outlier Analysis.* The data shows high consistency among the 18 participants, with one notable
 994 exception:

- 996 • **The Outlier (P03):** Participant P03 reported a “Moderate” workload (Mean: **4.33**), rating several dimensions at
 997 5/10. This deviation suggests that while the feature is intuitive for the vast majority, a small subset of users may
 998 lack specific prerequisite knowledge or experience an edge-case interaction.
- 1000 • **Consistency:** When excluding P03, the mean workload for the remaining 17 participants drops to **1.86**.
 1001 Furthermore, 94% of users (17/18) reported an overall workload below 3.0. This strong consistency statistically
 1002 validates the Reuse Assistant as a low-load feature for the target population.

1004 5 Discussion

1006 This study evaluated the Reuse Assistant, an automated guidance feature designed to help end-users recognize and
 1007 implement code reuse in block-based programming environments. Through a crossover study with 10 participants
 1008 from the chemistry domain, we assessed the feature’s impact on performance (RQ1), usability (RQ2), and perceived
 1009 workload (RQ3). The findings reveal both the potential and limitations of automated assistance in promoting software
 1010 reuse practices among domain experts with limited programming expertise.

1013 5.1 Implications for Theory

1015 5.1.1 *Reducing Attention Investment through Proactive Assistance.* Our study provides empirical support for the Attention
 1016 Investment Model [4] as applied to end-user development tools. The Attention Investment Model states that users make
 1017 rational decisions about whether to adopt new tools or features based on a cost-benefit analysis of the attention they
 1018 must invest upfront versus the perceived benefits they expect to receive, as well as the risk of there being no payoff at
 1019 all [5]. The higher the upfront attention cost (learning curve, discovery effort, comprehension requirements), the less
 1020 likely users are to adopt and utilize available features, even when those features would ultimately benefit their work.

1023 In the standard OpenRoberta environment, creating reusable custom blocks requires users to: (1) recognize that code
 1024 duplication exists and represents an opportunity for abstraction, (2) discover that custom block functionality is available
 1025 in the system, (3) locate where this feature resides in the interface, (4) understand how to use the feature correctly, and
 1026 (5) manually configure the custom block with appropriate parameters. This multi-step process represents a large upfront
 1027 attention investment that end-users, focused on their primary domain tasks rather than software engineering practices,
 1028 are unwilling or unable to make. In this study the result was 0% adoption of reusable components while using the
 1029 standard OpenRoberta Lab, despite participants possessing the cognitive capacity to understand and use custom blocks
 1030 when guided to do so. The Reuse Assistant changes this attention investment equation by eliminating steps 1-4 entirely.
 1031 Users do not need to recognize duplication patterns (the system detects them automatically), discover the feature
 1032 exists (the system actively presents opportunities), locate the feature in the interface (visual highlighting brings the
 1033 opportunity directly to users’ attention), or understand complex configuration procedures (automated parameterization
 1034 handles technical details). The upfront attention investment is reduced to a single decision: accept or reject the system’s
 1035 suggestion. This dramatic reduction in cognitive cost (from a complex multi-step learning process to a binary choice)
 1036 explains the 100% adoption rate in the Enhanced condition.

1041 Our findings extend the Attention Investment Model [4] by demonstrating that proactive, context-aware assistance
 1042 can transform feature adoption from an investment decision into an opportunistic choice. Rather than requiring users
 1043 to invest attention before experiencing any benefit, the Reuse Assistant delivers immediate, real value (highlighted
 1044 duplicates, one-click refactoring) that users can evaluate in real-time within their workflow. This "zero-cost trial"
 1045 approach eliminates the adoption barrier built-in to traditional feature-discovery models, where users must commit
 1046 attention resources before knowing whether the investment will prove worthwhile [5].
 1047

1048
 1049 *5.1.2 Addressing the Selection Barrier in End-User Development.* The results provide empirical evidence for a critical
 1050 distinction between different types of barriers to software reuse within Ko et al.'s [7] learning barriers framework. Among
 1051 the six barriers identified by Ko and colleagues (design, selection, coordination, use, understanding, and information
 1052 barriers), our work specifically addresses the *selection barrier*: the difficulty users face in knowing where to look for
 1053 features and choosing appropriate tools from the available options.
 1054

1055 The selection barrier appears in two distinct ways in block-based programming environments [7]. First, users must
 1056 know that reuse mechanisms exist and where to find them within the interface. Second, even when aware of available
 1057 features, users must determine when and how to apply them appropriately. Our 0% adoption rate in the standard
 1058 OpenRoberta condition demonstrates that the selection barrier is difficult to overcome for domain experts without
 1059 programming backgrounds, even when the interface provides the necessary functionality. Participants did not lack the
 1060 capability to create custom blocks (the same individuals achieved 100% adoption in the Enhanced condition) but rather
 1061 lacked the knowledge of where to look for this feature and when to apply it.
 1062

1063 The Reuse Assistant eliminates the selection barrier through two supporting mechanisms. First, automated detection
 1064 makes the feature location irrelevant. Users do not need to search the interface because the system proactively brings
 1065 the functionality to their attention at the appropriate moment. Second, context-aware suggestions eliminate the decision
 1066 burden about when to apply reuse. The system identifies appropriate opportunities and presents them when relevant,
 1067 allowing users to focus on domain-level acceptance decisions rather than technical feature selection.
 1068

1069 This finding extends Ko et al.'s framework by demonstrating that in block-based environments targeting end-users,
 1070 the selection barrier comes before and is more important than other barriers. The low NASA-TLX workload scores
 1071 (mean: 1.92) and high SUS scores (mean: 84.03) indicate that once the selection barrier is removed, once users no longer
 1072 need to find and choose features, the remaining barriers (use, understanding, coordination) impose minimal mental
 1073 burden. This suggests that tool designers should prioritize eliminating selection barriers through proactive assistance
 1074 before addressing other barrier types, as the latter become manageable once users are successfully guided to appropriate
 1075 features.
 1076

1077
 1078 *Relationship Between Recognition and Selection Barriers.* While Ko et al.'s selection barrier focuses on knowing where
 1079 to look for features, our work identifies a related but distinct *recognition barrier* specific to code reuse: users' inability
 1080 to identify opportunities for abstraction within their own code. These barriers are complementary. Even if users know
 1081 where the custom block feature is located (overcoming the selection barrier), they cannot use it effectively without
 1082 recognizing when their code contains patterns suitable for abstraction (overcoming the recognition barrier). Our Reuse
 1083 Assistant addresses both barriers simultaneously through automated pattern detection (recognition) and proactive
 1084 presentation (selection), explaining the dramatic shift from 0% to 100% adoption.
 1085

1086
 1087 *5.1.3 The Order Effect: Prior Experience as a Prerequisite for Appreciating Automation.* The significant order effect
 1088 ($t=-4.37$, $p=.008$) reveals a counter-intuitive finding: participants who received automated assistance first were actually
 1089

1093 1094 1095 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.

1096 1097 1098 1099 1100 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.

1101 1102 1103 1104 1105 This aligns with theories of learning transfer and expertise development [7], 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.

1106 1107 5.2 Implications for Practice

1108 1109 1110 1111 1112 1113 1114 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.

1115 1116 1117 1118 1119 1120 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.

1121 1122 1123 1124 1125 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.

1126 1127 1128 1129 1130 1131 1132 1133 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.

1134 1135 1136 1137 5.2.3 *Strategic Introduction of Automation.* The order effect findings have direct implications for training and onboarding. Organizations introducing automated coding assistants should consider implementing a staged approach:

- 1138 1139 1140 1141 1142 1143 1144 (1) **Initial Exposure Phase:** Allow users to complete initial tasks without automated assistance, building experiential understanding of manual processes and their pain points.
- 1140 1141 1142 (2) **Guided Automation Phase:** Introduce automated suggestions after users have established baseline workflows, ensuring they can appreciate what the automation provides.
- 1143 1144 (3) **Full Automation Phase:** Enable all automation features once users have developed adequate mental models.

1145 This staged approach goes against the intuitive "make it easy from the start" philosophy but may lead to better
 1146 long-term adoption and appropriate use of automation features.
 1147

1148 5.3 Threats to Validity

1150 5.3.1 Internal Validity.

1152 *Carryover effect.* While the within subjects design allowed within-subjects comparison, the significant carryover
 1153 effect ($p < 0.001$) indicates that the sequence of conditions fundamentally altered the user experience. This carryover
 1154 effect means we cannot cleanly separate the impact of the Reuse Assistant from the impact of prior experience. The
 1155 417 seconds performance gap between the two groups' average time differences suggests that learning from the first
 1156 condition substantially influenced performance in the second condition.
 1157

1158 *Mitigation:* We explicitly analyzed and reported the carryover effect as a finding rather than treating it as unwanted
 1159 noise. Furthermore, the experiments were balanced, meaning half of the participants performed the task with the Reuse
 1160 Assistant first, then again using the standard OpenRoberta. The other half did the reverse sequence. In this way, the
 1161 overall effect on the results is minimized.
 1162

1164 5.3.2 External Validity.

1166 *Convenience Sampling and Population Representation.* Participants were recruited through the researchers' professional
 1167 networks, creating a convenience sample. This introduces several limitations:
 1168

- 1169 • **Geographic and institutional diversity:** While the study included participants from multiple countries
 1170 (both local and international participants connected online), recruitment relied primarily on the researchers'
 1171 professional networks, which may not represent the full geographic and cultural diversity of potential end-users
 1172 in the domain of chemistry.
 1173
- 1174 • **Domain representation:** While participants came from diverse scientific backgrounds (chemistry, agronomy,
 1175 biochemistry) united by laboratory coursework experience, they represent primarily academic contexts rather
 1176 than industrial laboratory settings where cobot programming would be used professionally.
 1177
- 1178 • **Sample size:** With 18 participants for performance evaluation, usability assessment and workload assessment,
 1179 the study lacks statistical power to detect small effects or to adequately characterize rare user profiles(outliers),
 1180 limiting the generalizability of findings to broader populations.
 1181

1182 *Implications:* Findings should be interpreted as preliminary evidence rather than final proof of effectiveness across
 1183 all end-user developer populations. Replication studies with larger, more diverse samples from multiple institutions and
 1184 countries are necessary to establish the robustness of these results.
 1185

1186 *Ecological Validity: Laboratory vs. Authentic Use.* The study was conducted in a controlled setting with researcher
 1187 guidance available, tasks completed in a single session, and no real-world consequences for errors. This differs from
 1188 authentic usage where:
 1189

- 1190 • Users work independently without expert support
- 1191 • Programming tasks span multiple sessions with interruptions
- 1192 • Errors in cobot programs could damage equipment or compromise experiments
- 1193 • Users balance programming with their primary professional responsibilities

1197 *Mitigation:* We included chemistry domain experts as participants rather than generic users, and the task was based
1198 on actual laboratory procedures. However, long-term field studies observing the Reuse Assistant in authentic work
1199 contexts are necessary to validate its practical impact.
1200

1201 **5.3.3 Construct Validity.**
1202

1203 *Measurement Instruments.* We used standardized instruments (SUS questionnaire, NASA-TLX questionnaire) which
1204 have established validity in usability and cognitive workload research. However:
1205

- 1206 • **SUS limitation:** SUS assesses subjective usability perception rather than objective usability metrics, such as
1207 error rates or task success beyond completion time.
1208
- 1209 • **NASA-TLX limitation:** NASA-TLX assesses subjective workload perception, which may not correlate perfectly
1210 with objective cognitive load or learning outcomes.
1211

1212 **6 Conclusion and Future Work**
1213

1214 This study examined whether automated guidance can help end-users recognize and apply code reuse in block-based
1215 programming. We developed the Reuse Assistant, a tool that automatically detects duplicate code sequences and guides
1216 users to create reusable custom blocks in the OpenRoberta environment.
1217

1218 The results showed a clear difference in reuse adoption. While no participants created reusable blocks in the standard
1219 environment, all participants successfully created reusable blocks when help from the Reuse Assistant was available.
1220 The feature received high usability ratings (SUS mean: 84.03) and low workload scores (NASA-TLX mean: 1.92),
1221 demonstrating that automated guidance can be both effective and easy to use.
1222

1223 Our findings contribute to theory by extending the Attention Investment Model and the Learning Barriers Framework.
1224 We showed that proactive assistance reduces the upfront cost of adopting new features and that the selection barrier
1225 is particularly important in block-based environments for end-users. The significant order effect revealed that prior
1226 manual experience helps users appreciate automation benefits.
1227

1228 For practice, this study demonstrates that simple design choices matter. Visual highlighting, one-click acceptance,
1229 and immediate feedback were sufficient to achieve high adoption of reusable blocks without adding complexity. The
1230 results suggest that programming environments for domain experts should actively guide users rather than waiting for
1231 them to discover features independently.
1232

1233 **6.1 Future Work**
1234

1235 Future research should test the Reuse Assistant in real laboratory settings over extended periods to determine whether
1236 users learn to recognize reuse opportunities independently. Studies with more participants from diverse backgrounds
1237 would help identify which user groups benefit most from automated guidance. The feature should also be evaluated
1238 with more complex programming tasks and tested in other end-user programming environments beyond OpenRoberta.
1239

1240 **References**
1241

- [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.

1242 Manuscript submitted to ACM
1243

- 1249 [4] Alan F. Blackwell. 2002. First Steps in Programming: A Rationale for Attention Investment Models. In *Proceedings of the IEEE Symposia on Human*
 1250 *Centric Computing Languages and Environments*. IEEE Computer Society, 2–10. doi:10.1109/HCC.2002.1046334
- 1251 [5] Alan F. Blackwell and Thomas R. G. Green. 2003. Notational Systems - The Cognitive Dimensions of Notations Framework. *HCI Models, Theories,*
 1252 *and Frameworks: Toward a Multidisciplinary Science* (2003), 103–133.
- 1253 [6] Alexander Bock and Ulrich Frank. 2021. Low-Code Platform. *Business and Information Systems Engineering* 63 (2021). doi:10.1007/s12599-021-00726-8
- 1254 [7] Andrew J. Ko, Brad A. Myers, and Htet Htet Aung. 2004. Six Learning Barriers in End-User Programming Systems. *IEEE Symposium on Visual*
 1255 *Languages and Human-Centric Computing (VL/HCC)* (2004), 199–206. doi:10.1109/VLHCC.2004.47
- 1256 [8] Yuhan Lin and David Weintrop. 2021. The Landscape of Block-Based Programming: Characteristics of Block-Based Environments and How They
 1257 Support the Transition to Text-Based Programming. *Journal of Computer Languages* 67 (2021), 101075. doi:10.1016/j.cola.2021.101075
- 1258 [9] Hugo Lourenço, Carla Ferreira, and João Costa Seco. 2021. OSTRICH - A Type-Safe Template Language for Low-Code Development. In *2021 ACM/IEEE*
 1259 *24th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. 216–226. doi:10.1109/MODELS50736.2021.00030
- 1260 [10] Vlad Magdalin. 2012. Low code platform tool Webflow. <https://webflow.com/>.
- 1261 [11] Derek Roos. 2005. Low code platform tool Mendix. <https://www.mendix.com/>.
- 1262 [12] Roel J. Wieringa. 2014. *Design Science Methodology for Information Systems and Software Engineering*. Springer, Berlin, Heidelberg. doi:10.1007/978-
 3-662-43839-8

1264 A System Usability Scale (SUS) Questionnaire

1265 The System Usability Scale (SUS) is a widely used standardized questionnaire for assessing the perceived usability of
 1266 a system. Participants respond to each statement using a 5-point Likert scale ranging from 1 (Strongly Disagree) to
 1267 5 (Strongly Agree). The SUS score is calculated by converting the responses to a scale of 0-100, where higher scores
 1268 indicate better usability.

1271 A.1 SUS Statements

- 1272 (1) I think that I would like to use the Reuse Assistant feature frequently.
 1273 (2) I found the Reuse Assistant feature unnecessarily complex.
 1274 (3) I thought the Reuse Assistant feature was easy to use.
 1275 (4) I think that I would need the support of a technical person to be able to use the Reuse Assistant feature.
 1276 (5) I found the various functions in the Reuse Assistant feature were well integrated.
 1277 (6) I thought there was too much inconsistency in the Reuse Assistant feature.
 1278 (7) I would imagine that most people would learn to use the Reuse Assistant feature very quickly.
 1279 (8) I found the Reuse Assistant feature very cumbersome to use.
 1280 (9) I felt very confident using the Reuse Assistant feature.
 1281 (10) I needed to learn a lot of things before I could get going with the Reuse Assistant feature.

1286 A.2 Scoring Method

1287 For odd-numbered items (1, 3, 5, 7, 9), the score contribution is the scale position minus 1. For even-numbered items (2,
 1288 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
 1289 obtain the overall SUS score, which ranges from 0 to 100.