

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

3 ANNE-MARIE ROMMERDAHL, SDU, Denmark

4 JEREMY ALEXANDER RAMÍREZ GALEOTTI, SDU, Denmark

5 DIMITRIOS DAFNIS, SDU, Denmark

6 NASIFA AKTER, SDU, Denmark

7 MOHAMMAD HOSEIN KARDOUNI, SDU, Denmark

8 **Abstract**—End-users who program collaborative robots for laboratory automation often create repetitive code because
9 they struggle to recognize opportunities for reuse. While block-based programming environments provide accessible
10 interfaces, they do not actively guide end-users toward creating reusable components. This study investigates whether
11 automated guidance can help end-users recognize and apply code reuse practices. We developed the Reuse Assistant, a
12 feature that automatically detects duplicate code sequences within the OpenRoberta environment and guides users to
13 create reusable custom blocks through visual highlighting and one-click refactoring. Through a within-subjects study with
14 18 participants from the chemistry domain, we evaluated the feature’s impact on performance, usability, and perceived
15 workload. Automated guidance increased reuse adoption from 0% in the standard OpenRoberta version to 100% when
16 using the Reuse Assistant. The feature achieved high usability scores (SUS mean: 84.03) and imposed minimal cognitive
17 burden (NASA-TLX mean score: 1.92). The significant order effect revealed that prior manual experience helps users
18 appreciate automation benefits. This dramatic shift in adoption suggests that end-users are capable of using advanced
19 features if the system actively guides them.

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

21 **ACM Reference Format:**

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

26 **1 Introduction**

27 Software reuse is a fundamental practice in software engineering, enabling developers to build on existing solutions
28 rather than writing code from scratch. However, end-users who program collaborative robots (cobots) for laboratory
29 automation often lack the knowledge to recognize and apply reuse opportunities. This problem is particularly acute

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

33 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
34 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
35 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
36 servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

37 © 2018 Copyright held by the owner/author(s). Publication rights licensed to ACM.

38 Manuscript submitted to ACM

39 Manuscript submitted to ACM

in domains like chemistry, where scientists need to automate repetitive experimental procedures but have limited programming expertise.

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

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

Our investigation examined four research questions:

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

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

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

2 Background and Related Work

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

A study by Bock and Frank (2021) studied several low-code platforms (LCPs), in order to identify characteristic features of LCPs. The identified features were presented according to how frequently they 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[6]. This lack of focus on promoting reuse may impact the so-called 'Citizen Developers', who have little or no coding knowledge, and whom may then miss out on the benefits of reuse. Lin and Weintrop (2021) noted that most existing research on block-based programming focuses on supporting the transition to text-based languages rather than exploring how features within BBP environments, such as abstraction or reuse, can enhance learning outcomes[11].

105 There have been proposed some ideas on how to promote reuse for LCPs, such as the templating language OSTRICH,
 106 developed for the model-driven low-code platform OutSystems[12]. OSTRICH is designed to assist the end-user in
 107 making use of OutSystems' available templates, by abstracting and parameterizing the templates. However, OSTRICH
 108 only supports the top nine most used production-ready screen templates, and does not allow the end-user to create and
 109 save their own templates, or re-apply a template which they have customized. Another approach focused on enabling
 110 the reuse of models, by providing recommendations to the end-user, based on the models stored in a graph acting as
 111 a repository. While the graph allows end-users to reuse their own models, there is no mention of guiding the user
 112 towards reusing their own models.
 113

114 Several popular low-code development platforms (LCDPs) provide different kinds of support for reuse. Webflow[13],
 115 a LCDP for responsive websites, offers the ability to create reusable components and UI kits, which can be reused across
 116 multiple pages and projects. Mendix[14] and OutSystems offer even more functionality to support reuse, offering several
 117 ways to end-users to share their code with each other, and offering pre-made components. Both of these platforms also
 118 utilize AI to enhance reuse. Outsystems provides AI suggestions to spot and create reusable pieces, while Mendix uses
 119 AI to suggest the best solutions and components for specific tasks. However, well-known pitfalls of AI are its tendency
 120 to generate non-deterministic outputs, and hallucinations. While an experienced programmer can critically analyze the
 121 output of the AI, the common end-user lacks that ability. In order to analyze how block-based robotics environments
 122 address reuse, 4 representative platforms were compared: mBlock, MakeCode, SPIKE LEGO, VEXcode GO and Open
 123 Roberta. The comparison focused on three main dimensions of reuse: structural reuse (through user-defined blocks or
 124 functions), social reuse (through sharing or remixing existing projects), and interoperable reuse (through import/export
 125 capabilities).
 126

127

128 **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

141

142 In this context, “reuse support” represents a scale that measures how effectively each platform facilitates reuse-related
 143 features. High reuse support indicates that users can easily create, share, and adapt existing components or projects.
 144 Medium reuse support suggests that some reuse mechanisms are available but limited in scope or flexibility. Low reuse
 145 support implies that the platform provides only minimal or restricted features to promote reuse.
 146

147 As shown in Table 1, although these platforms include reusability features, they are quite limited, as none of them
 148 provide users with clear guidance on how to use these tools effectively, which restricts their ability to fully leverage them.
 149

150 A study by Techapalokul and Tilevich (2019) suggests that supporting mechanisms for reusing smaller, modular
 151 pieces of code can enhance programmer productivity, creativity and learning outcomes. Adler et al. (2021) introduced a
 152 search-based refactoring approach to improve the readability of Scratch programs by automatically applying small code
 153 154 155 156

157 transformations, such as simplifying control structures and splitting long scripts[1]. Their findings demonstrated that
158 automated refactoring can significantly enhance code quality and readability for novice programmers.
159

160 Building upon all these concepts and ideas, our project introduces a guided Reuse Assistant within the OpenRoberta
161 Lab environment. The tool is designed to help users identify and apply reuse more easily while creating their robot
162 programs. Focused on guiding users toward creating reusable custom blocks to promote modularity and abstraction,
163 the tool automatically scans a user's block-based program to detect repeated code segments in the workspace. The
164 system visually highlights the found duplicates, drawing the user's attention to patterns that can be reused. The tool
165 also offers the functionality to create the custom block for the end-user, by identifying the small differences between
166 the repeated parts (such as numbers, variables, or parameters) and turning these differences into inputs for the new
167 block. The tool automatically replaces all relevant duplicate sequences with the new custom block.
168

169 By combining ideas from procedural abstraction (organizing code into meaningful, reusable parts) and automated
170 refactoring (improving code through intelligent transformations), our tool aims to make block-based programming
171 more structured and efficient. It encourages users to build programs that are modular and easier to maintain, helps
172 reduce unnecessary repetition, and supports learning by making the concept of reuse clear and hands-on.
173

175 3 Study Design

177 Following the Design Science methodology [16], the study is structured into three main phases: problem investigation
178 to define goals, treatment design to specify the artifact requirements, and treatment validation to assess the artifact's
179 performance in a controlled environment.
180

182 3.1 Problem Investigation

184 3.1.1 *Problem Context and Motivation.* End-user development (EUD) for collaborative robots (cobots) presents unique
185 challenges, particularly for users without formal programming training. In domains such as chemistry, educational
186 robotics, and industrial settings, end-users need to program robots to perform specific tasks but often lack the software
187 engineering knowledge to write maintainable, well-structured code. In the domain of Chemistry, one of the most
188 relevant and important tasks is performing experiments in labs. Robots can be used in chemistry labs to automate
189 experiments with great effect, as many experiments involve steps that are repetitive, and susceptible to human error,
190 such as a step being overlooked, instructions being misread, etc. Automation of menial tasks will leave the chemists
191 with more time for other work, with the added bonus of not having to handle dangerous chemicals.
192

194 One critical challenge in EUD is code reuse. Users frequently create repetitive code as they struggle to recognize
195 duplicate patterns, lack knowledge about abstraction mechanisms, or find existing tools too complex to use effectively.
196 This problem manifests in several ways: programs become unnecessarily long and difficult to maintain and small
197 changes require modifications in multiple locations, increasing the risk of errors. So, while the use of robots in chemistry
198 lab work offer great benefits, the challenge of automating the repetitive work may turn chemists away from using
199 robots.
200

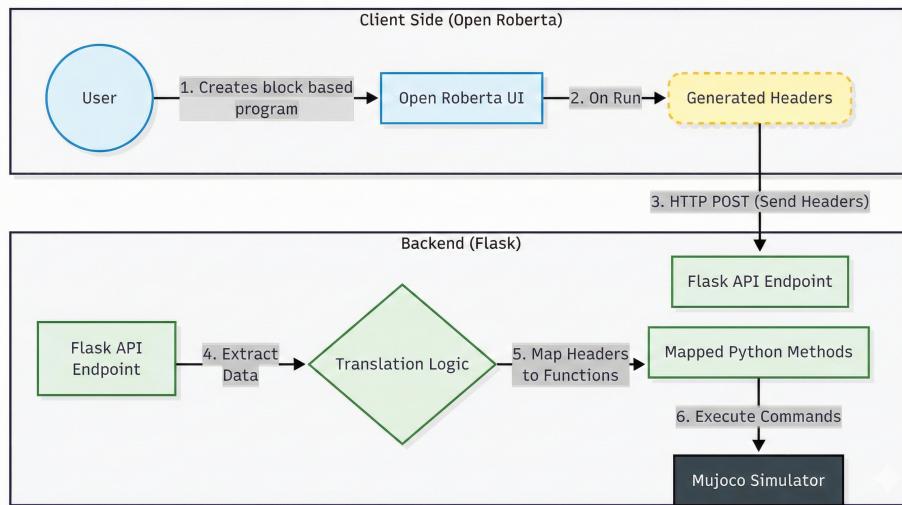
202 3.1.2 *Stakeholder Analysis.* Chemists and lab technicians who use cobots for repetitive tasks such as sample preparation,
203 mixing, quality control procedures, etc. They possess deep domain expertise in chemistry but limited programming
204 knowledge, often creating long, repetitive programs that become difficult to maintain when adapting experimental
205 protocols. Their primary need is to quickly create and modify robot programs without becoming programming experts.
206

209 **3.2 Treatment Design**

210 To address the problem of code reuse in EUD for cobots, we have derived a set of requirements designed to contribute
 211 to the chemist's goal of creating maintainable and reusable robot programs. Functionally, the artifact must be capable
 212 of automatically detecting duplicate or similar block sequences and visually highlighting these duplications within the
 213 user's workspace. These requirements are necessary to help the end-user recognize opportunities for reuse, that would
 214 otherwise go unnoticed. Once detected, the system must offer to create reusable custom blocks, allowing the user to
 215 accept or reject these suggestions. These signals are important, as they give the end-user control over the reuse process,
 216 allowing them to decide when and how to apply reuse in their programs. Regarding non-functional requirements, the
 217 artifact must seamlessly integrate with the existing Open Roberta Lab environment to ensure a smooth user experience.
 218 The interface should be intuitive for end-users, minimizing the learning curve and making it easy to understand and use
 219 the reuse features. Additionally, the artifact should not interfere with the existing workflow, allowing users to continue
 220 their programming tasks without disruption. To satisfy the requirements above, we designed the Reuse Assistant as a
 221 feature for the Open Roberta Lab.

222 **3.2.1 Architecture.** The system enables the execution of block-based programs on a simulated cobot through a three-tier
 223 architecture, as illustrated in figure 1. The workflow consists of the following stages:

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



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

261 3.2.2 *Detection Algorithm.* The algorithm follows three main steps:

- 262** • **Linearization:** First, the algorithm linearizes the block workspace into a sequential list of blocks.
- 263** • **Identify sequences:** It iterates through this list to identify all possible sequences of blocks that meet a minimum
- 264** unique block type length requirement (three blocks).
- 265** • **Sequences Matching:** If the same sequence of block types is found more than once, it will be added to the
- 266** ReusableComponentCandidates list, which will be sorted by longest and most recent duplicated sequences. In
- 267** the end the highest priority candidate gets returned.
- 268**

269 The pseudocode is illustrated in Algorithm 1.

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

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

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

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

```

274 1: Chain = buildLinearChain(StartBlock)
275 2: Sequences = List<sequence>
276 3: for startIndex = 0 to length(Chain) - 1 do
277 4:   for sequenceLength = 1 to MaxSequenceLength do
278 5:     sequence = Chain[startIndex .. startIndex + sequenceLength - 1]
279 6:     numberofBlockTypesInSequence = getNumberOfDistinctBlockTypes(sequence)
280 7:     if sequenceLength >= MinimumSequenceLength and numberofBlockTypesInSequence >= MinimumDifferentBlockTypesInSequence then
281 8:       Sequences.append(sequence) // record sequence occurrence
282 9:     end if
283 10:   end for
284 11: end for
285 12: ReusableComponentCandidates = {Sequences | occurrence ≥ 2}
286 13: sort ReusableComponentCandidates by (longest sequence length and most recent occurrence)
287 14: return ReusableComponentCandidates[0] // Return highest priority candidate

```

293 3.2.3 *User Interface and Interaction.* The user interface is designed to be intuitive and non-disruptive. When the

294 detection algorithm identifies a candidate, the system visually highlights the relevant blocks on the canvas as shown

295 in Figure 2. A non-blocking toast notification appears, prompting the user to confirm the refactoring. If confirmed,

296 the system automatically generates the custom block definition in a dedicated workspace area and updates the main

297 workspace, replacing the redundant code with concise function calls as shown in Figure 3. This process abstracts the

298 complexity of manual function creation, guiding the user toward modular design practices. After the user presses the

299 run simulation button, the robot simulator of mujoco opens up and executes the commands provided by the user inside

300 the Open Roberta workspace. This is illustrated in Figure 4.

301

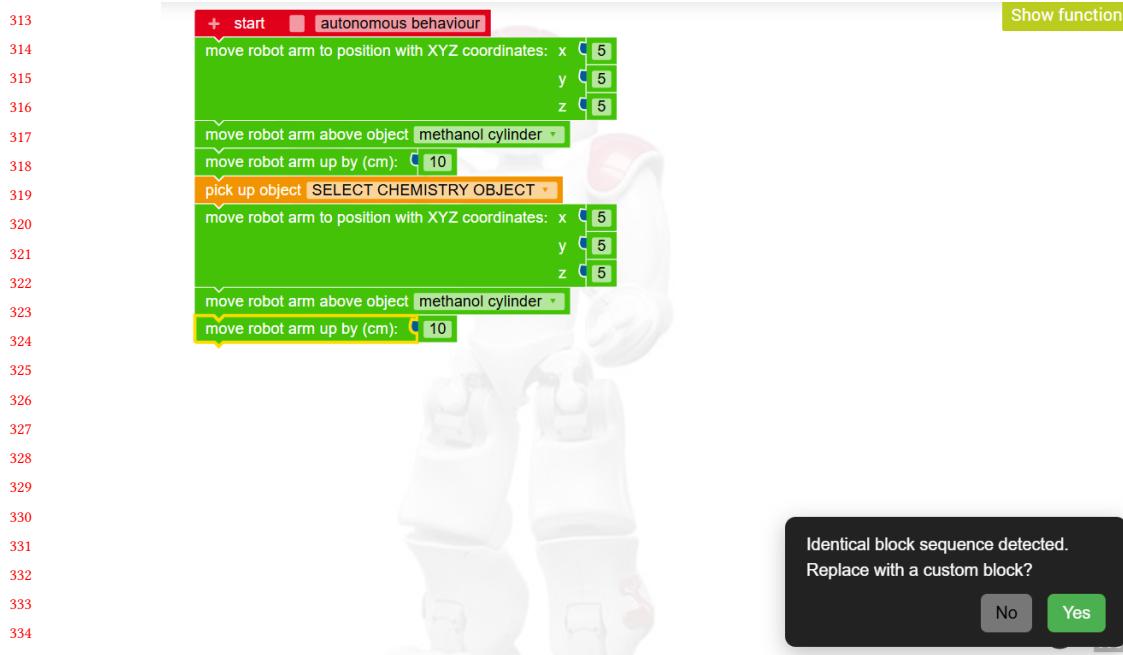


Fig. 2. Reuse Assistant workflow: detection - the interface highlights detected duplicate blocks by changing their color to green.

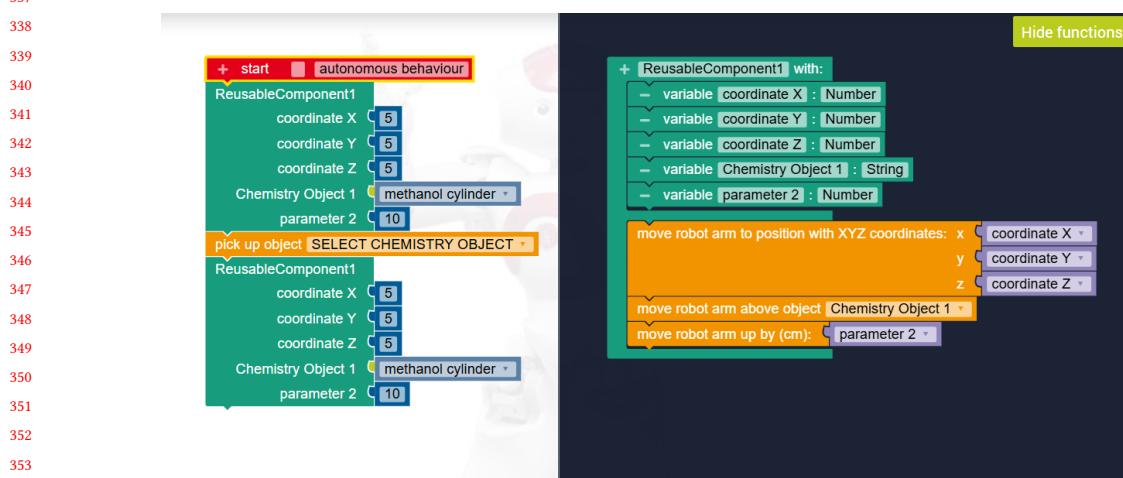


Fig. 3. Reuse Assistant workflow: refactoring - the automated refactoring result, showing the new custom block definition and the simplified main program.

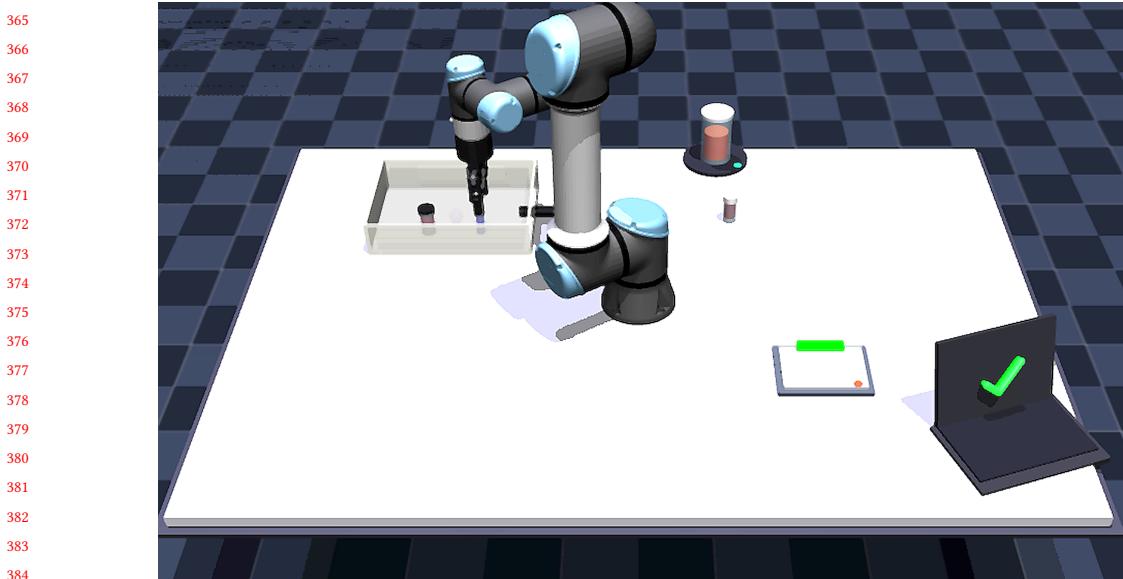


Fig. 4. Mujoco robot simulator executing the commands from Open Roberta.

3.3 Treatment Validation

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

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

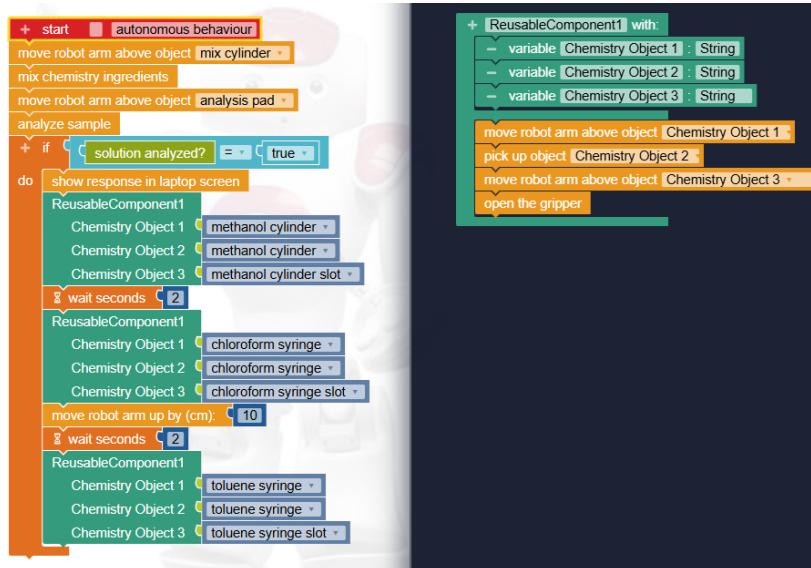
3.3.2 Task Execution. The participants were initially given a short introduction to the Open Roberta UI, as well as the mujoco robot simulator. They then performed one task which is described by a set of pre-defined steps. This task has been specifically designed to promote the reusability aspect. The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

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

- (1) Move the robot arm above mix cylinder
- (2) Mix the chemistry ingredients

- 417 (3) Move the robot arm above the analysis pad
 418 (4) Analyze the sample
 419 (5) If the solution is analyzed (use if statement) then show a response message in the laptop's screen
 420 (6) Place the following three objects into their corresponding slots in the chemistry equipment toolbox:
 421 • Methanol cylinder
 422 • Chloroform syringe
 423 • Toluene syringe
 424 (7) Important notes for the participants:
 425 • After placing an object to its slot in the toolbox **wait 2 seconds** before you move to pick a new one.
 426 • 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
 427 • Click the **play** button on the bottom right corner to start the simulation
 428 • Click the **reset** button on the bottom right corner to reset the scene of the robot simulator

429 The most optimal solution as defined by the researchers is illustrated in Figure 5. Instead of creating a long linear



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

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

434 All the participants will perform the task using both the enhanced version of Open Roberta, which includes the Reuse
 435 Assistant, and the original one which does not. The study employs a within-subjects design, where all participants
 436 are divided into two groups. Half of the participants (Group A) will first perform the task using the enhanced version,
 437 then the original version. The other participants (Group B) will do the opposite order. Participants' interactions with
 438

469 the platform will be observed throughout the task. Guidance will be provided from the researchers to the participants
470 throughout the task.
471

472 *3.3.3 Data Gathering and Analysis.* The data collection focused on numerical measurements of performance and
473 participant feedback:
474

- 475 (1) **Task Completion Time:** Measured in seconds for both versions of Open Roberta (enhanced and original) to
476 evaluate efficiency gains. Statistical analysis employed paired t-tests to assess within-group improvements and
477 a Welch's t-test to compare improvement scores between groups, specifically isolating the impact of the order
478 effect arising from the sequence of conditions.
479
- 480 (2) **Reuse adoption and functional correctness:** Evaluated by tracking the voluntary implementation of reusable
481 custom blocks during the task. It was measured how many participants let the Reuse Assistant create custom
482 blocks for them. Functional correctness was assessed by verifying if the robot successfully executed the chemical
483 mixing protocol, with minor variations in safety steps deemed acceptable if they did not affect the simulation
484 outcome.
485
- 486 (3) **Usability Assessment:** Evaluated using the System Usability Scale (SUS) questionnaire to measure participants'
487 perceived usability of the Reuse Assistant feature. The SUS yields a single number representing a composite
488 measure of the overall usability of the system, with scores ranging from 0 to 100.
489
- 490 (4) **Workload Assessment:** Measured using the NASA post-task Workload questionnaire (NASA-TLX) to assess
491 the cognitive demands imposed by the Reuse Assistant across six dimensions (mental demand, physical demand,
492 temporal demand, performance, effort, and frustration).
493

494 This comprehensive evaluation provided a detailed understanding of how useful and effective the Reuse Assistant
495 feature is to the end-users.
496

497 4 Results

498

499 4.1 Research Question 1: How does the Reuse Assistant affect the end-users' performance?

500 To evaluate the impact of the Reuse Assistant on end-user performance, we measured task completion times for all
501 participants using either version (enhanced or original) of Open Roberta.
502

503 Tables 2 and 3 present the individual completion times for all participants using either version of Open Roberta.
504 The data reveals substantial variability in performance outcomes depending on the order in which the participants
505 performed the tasks, with Group B participants showing consistent improvements when moving from the original
506 Open Roberta version to the enhanced, while Group A participants exhibited the opposite pattern.
507

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

Table 2. Task Completion Times of group A

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

Table 3. Task Completion Times of group B

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

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

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

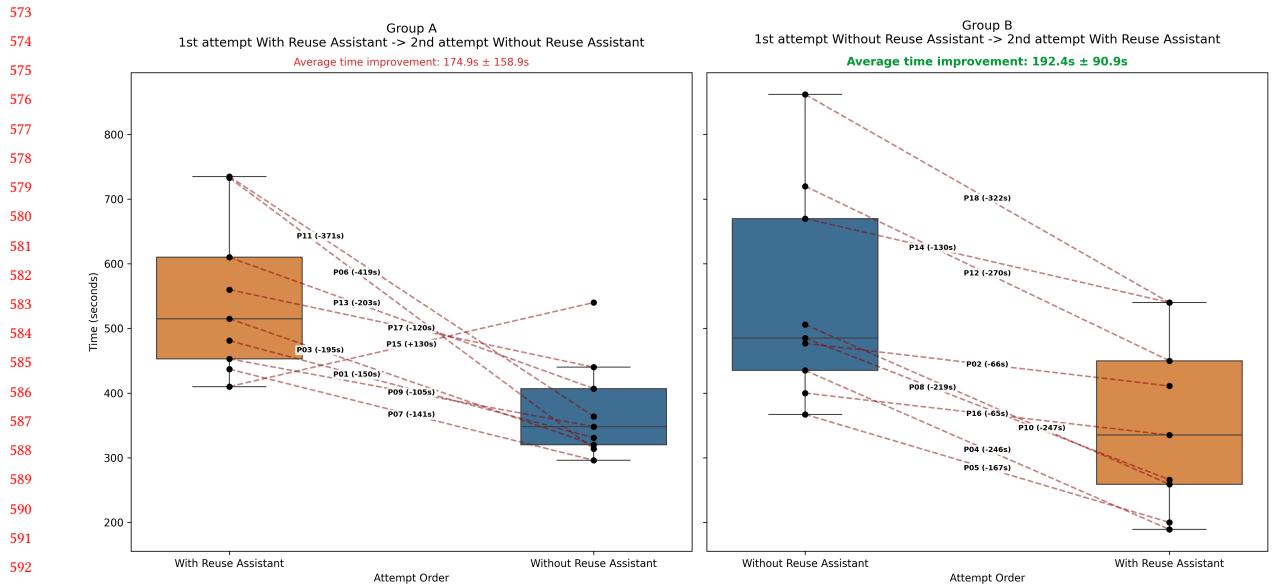


Fig. 6. Distribution of task completion times, comparing Group A and Group B participants across both versions of Open Roberta.

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

Table 4. Statistical Test Results

Overall Comparison. When combining all 18 participants, regardless of the order in which they experienced the two versions of Open Roberta, the overall mean time difference was -8.78 seconds (Standard Deviation = 226.90), leading to a t-value = -0.16 ($p = 0.872$). This non-significant result indicates no overall difference when order is ignored.

Group A Analysis. Group A first performed the task with the Reuse Assistant available, and then without it. The mean difference (Completion time with Reuse Assistant - Completion time without Reuse Assistant) was 174.9 seconds (Standard Deviation = 158.9 seconds), producing a significant result with a t-value of 3.30 ($p = 0.011$). The positive value indicates that these participants were *faster* on their second attempt, where the Reuse Assistant was unavailable.

Group B Analysis. Group B first performed the task with the Reuse Assistant being unavailable, then again with it being available. We calculated the time difference (Completion time with Reuse Assistant - Completion time without Reuse Assistant) for each participant. The mean improvement was -192.44 seconds (SD = 90.91), yielding a t-value of -6.35 ($p < 0.001$). This statistically significant result demonstrates that these participants were a lot *faster* on their

625 second attempt, where the Reuse Assistant was available. The low standard deviation reveals that there was a consistent
 626 pattern in how much faster participants worked with the feature being available.
 627

628 *Order Effect Analysis.* To determine whether the order in which participants used the two versions influenced their
 629 performance, we conducted a Welch's t-test, comparing the improvement scores between Group A and Group B. This
 630 analysis revealed a highly significant order effect ($t = 6.02, p < 0.001$).
 631

632 The difference between the two groups was massive with a gap of 367 seconds. Group B participants, who started the
 633 task without the Reuse Assistant, finished 192 seconds faster once they had the Reuse Assistant available. In contrast,
 634 Group A participants started with the Reuse Assistant, but they actually took 175 seconds longer to finish compared to
 635 their subsequent performance in the unassisted condition.
 636

637 This big difference points to a strong learning effect. The results show that participants were faster mainly because
 638 they got used to the task, not just because of the benefits of the Reuse Assistant. It did not matter if they started with the
 639 Reuse Assistant or without it. The experience from the first try made the second attempt to solve the task much easier.
 640

641 4.1.2 *Summary of Findings.* The Reuse Assistant effectively helped the end users complete the task faster. We can
 642 conclude this by comparing the average time differences and standard deviations between the two groups of participants.
 643

644 Participants in Group B improved their time by an average of 192.4 seconds when they switched to using the feature.
 645 This is a larger change than Group A. Participants in Group A were faster by an average of 174.9 seconds when they
 646 switched to using the original version. This shows that the gain from adding the assistant was greater than the loss
 647 from removing it.
 648

649 The standard deviation also tells us about consistency. Group B had a low standard deviation of 90.9 seconds. In
 650 contrast, Group A had a much higher standard deviation of 158.9 seconds. This means that the performance boost was
 651 not only larger but also more consistent when users utilized the Reuse Assistant.
 652

653 Finally, the statistical strength confirms this result. The t-value for Group B is -6.35. This is much stronger than the
 654 t-value for Group A which is 3.30. Since the Group B result is statistically more significant, we can say that the Reuse
 655 Assistant provides a clear and robust performance advantage.
 656

657 4.2 Research Question 2: To what extent does the reuse assistant facilitate reusability?

658 *Adoption of Reusable Blocks.* In the Enhanced Open Roberta version that includes the Reuse Assistant feature, 18/18
 659 participants accepted the Reuse Assistant's offer to create a custom reusable block to handle the repetitive object
 660 placement steps. In contrast, in the *original* Open Roberta version, participants predominantly relied on linear, repetitive
 661 code structures. Without the guidance features, none of them attempted creating a reusable block.
 662

663 4.2.1 *Summary of Findings.* The Reuse Assistant promotes reusability by automating the manual effort required to
 664 build reusable blocks. It achieves this by lowering the cognitive and technical barriers associated with identifying and
 665 abstracting repetitive patterns.
 666

667 For both Group A and B, there were 0 out of 18 participants who created custom blocks while using the original Open
 668 Roberta. On the other hand, all 18 participants both created and used a custom block when the Reuse Assistant was
 669 available. Thus we can conclude that the Reuse Assistant successfully helped facilitate reuse whenever it was available
 670 to the participants.
 671

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

Data about perceived usability of the Reuse Assistant feature was collected by having all $N = 18$ participants answer the System Usability Scale (SUS) questionnaire (see Appendix A for the full questionnaire details) immediately after they had finished their two tasks.

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

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

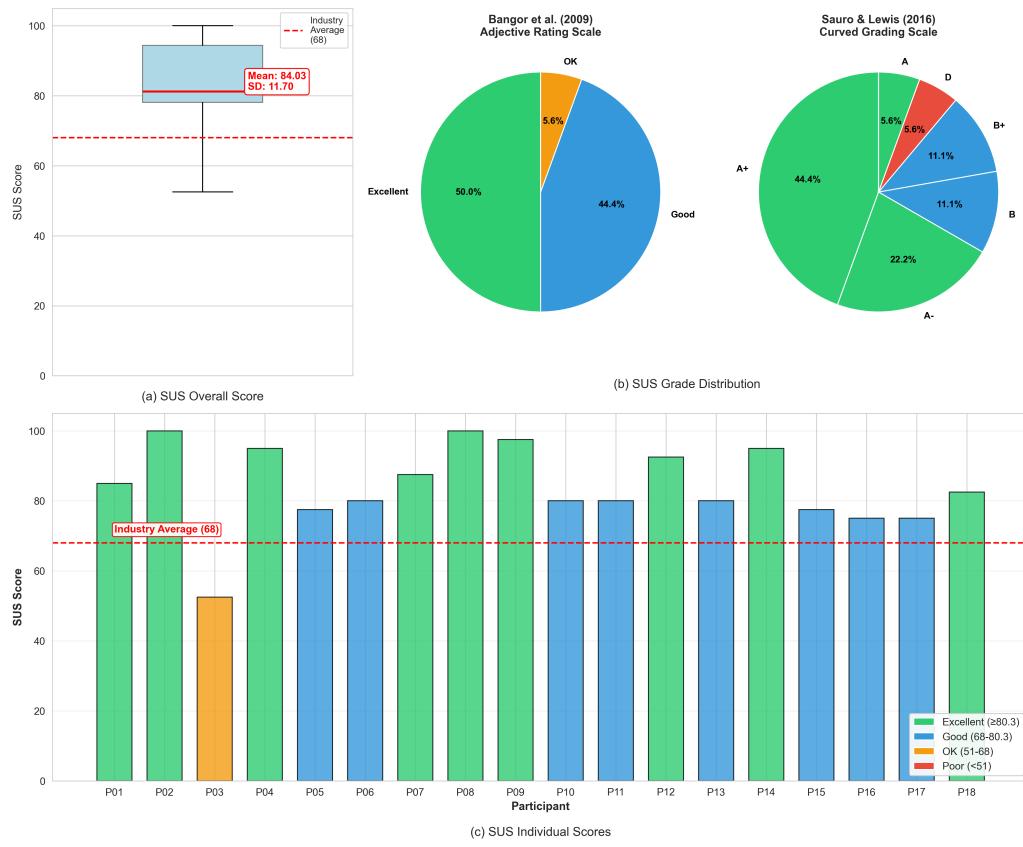


Fig. 7. System Usability Scale (SUS) Analysis Results. (a) Distribution of SUS Scores among Participants. (b) Dual classification showing Bangor et al.’s adjective ratings (left) and Sauro & Lewis letter grades (right).

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

Manuscript submitted to ACM

729 rated it as "Excellent" (85–100), and nine people rated it as "Very Good" (75–80). This suggests that almost everyone
 730 understood how to use the system right away.
 731

732 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
 733 scores very close to each other (between 75 and 100), which suggests the system is consistently easy to use for most
 734 people.
 735

736 *4.3.3 Summary of Findings.* End-users assess the Reuse Assistant as an exceptionally usable system, with a majority
 737 rating it "Excellent", and the scores showing a high degree of consistency.
 738

739 Firstly, users judge the system to be superior to the industry standard. They gave it an average score of 84.0 which
 740 is higher than the typical average of 68. This shows that users perceive the feature as significantly easier to use than
 741 standard software.
 742

743 Secondly, the scores show a strong consistency between users. 17 out of 18 participants rated the usability above 75.
 744 This tight clustering of scores proves that the positive experience was uniform across the group. Almost every user
 745 agreed that the feature was intuitive.
 746

747 Thirdly, users evaluate the system as easy to learn. The high scores indicate that participants felt confident operating
 748 the feature immediately. They did not struggle to understand how it worked. Therefore, users assess the Reuse Assistant
 749 as a polished and highly accessible solution.
 750

751 **4.4 Research Question 4: How do the end-users assess the perceived workload when operating the Reuse 752 Assistant?**

753 To assess the cognitive demands imposed by the Reuse Assistant, we administered the NASA Task Load Index (NASA-
 754 TLX) questionnaire (see Appendix B for the full questionnaire details) to all participants after completing the task with
 755 the Reuse Assistant benefits. The NASA-TLX is a widely used multidimensional assessment questionnaire that measures
 756 perceived workload across six subscales, each rated on a scale from 1 (Very Low) to 10 (Very High). The scores were
 757 calculated by multiplying each rating by 10 to convert them to a 0-100 scale.
 758
 759

760 *4.4.1 Overall Workload Assessment.* As illustrated in figure 8, the Reuse Assistant imposes a remarkably low workload
 761 on users, with an overall mean score of **20.00**. This indicates that the feature is highly usable and demands very little
 762 effort from the user in terms of physical or mental cost.
 763

764 *4.4.2 Dimension-Specific Analysis.* There is a distinct contrast between the highest and lowest contributing factors:
 765

- 766 • **Lowest Demand (Temporal):** With a mean score of **10.6**, Temporal Demand was negligible. This confirms
 767 that users felt **zero time pressure**, allowing them to work at their own pace without stress.
 768
- 769 • **Highest Demand (Effort):** Effort received the highest rating (**30.00**), yet this is still considered "Low" on
 770 a 100-point scale. This suggests a positive trade-off: users were cognitively engaged (concentrating) but not
 771 overworked. The feature requires attention, but not exhaustion.
 772

773 *4.4.3 Consistency and Outlier Analysis.* The data shows high consistency among the 18 participants, with one notable
 774 exception:
 775

- 776 • **The Outlier (P03):** Participant P03 reported a "Moderate" workload (Mean: **43.3**), rating several dimensions at
 777 5/10. This deviation suggests that while the feature is intuitive for the vast majority, a small subset of users may
 778 lack specific prerequisite knowledge or experience an edge-case interaction.
 779

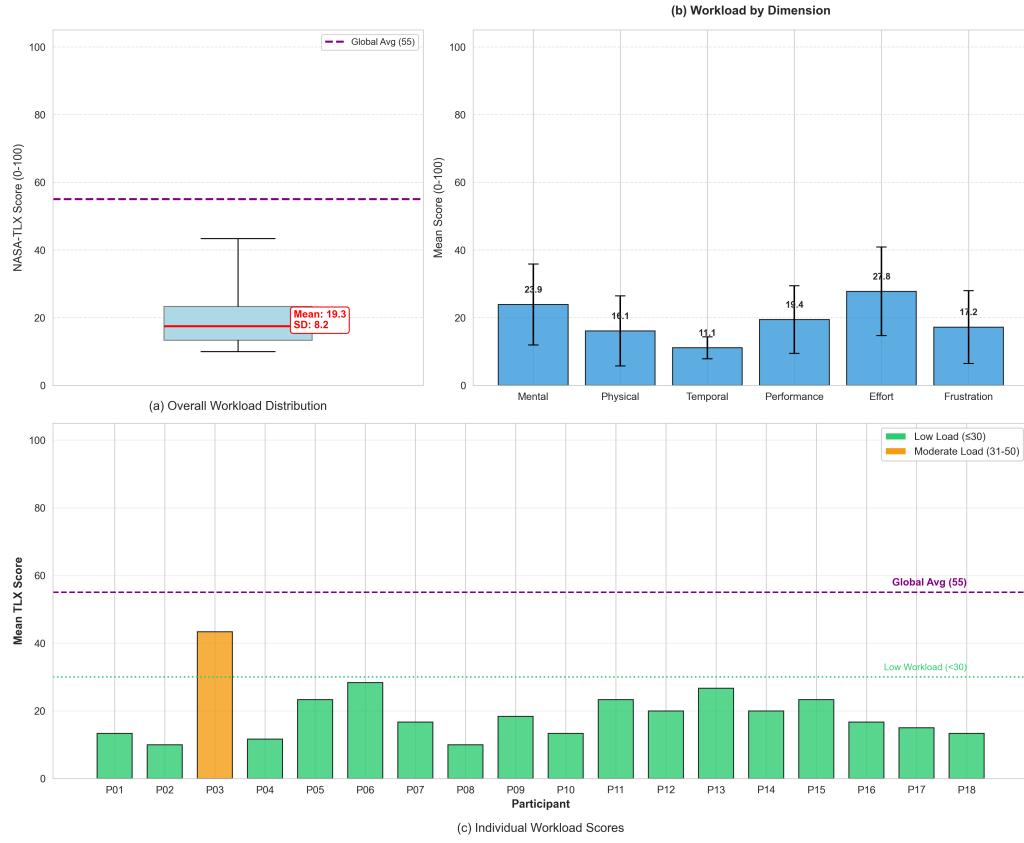


Fig. 8. NASA-tlx workload analysis results. (a) Distribution of NASA-TLX overall workload score (b) Mean workload scores for each dimension. (c) Distribution of workload scores across participants.

- **Consistency:** When excluding P03, the mean workload for the remaining 17 participants drops to **18.6**. Furthermore, 94% of users (17/18) reported an overall workload below 30.0. This strong consistency statistically validates the Reuse Assistant as a low-load feature for the target population.

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

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

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

833 5 Discussion

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

841 842 5.1 Implications for Theory

843 844 5.1.1 Reducing Attention Investment through Proactive Assistance. Our study provides empirical support for the Attention
 845 Investment Model [4] as applied to end-user development tools. The Attention Investment Model states that users make
 846 rational decisions about whether to adopt new tools or features based on a cost-benefit analysis of the attention they
 847 must invest upfront versus the perceived benefits they expect to receive, as well as the risk of there being no payoff at
 848 all [5]. The higher the upfront attention cost (learning curve, discovery effort, comprehension requirements), the less
 849 likely users are to adopt and utilize available features, even when those features would ultimately benefit their work.

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

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

870 871 5.1.2 Addressing the Selection Barrier in End-User Development. The results provide empirical evidence for a critical
 872 distinction between different types of barriers to software reuse within Ko et al.'s [10] learning barriers framework.
 873 Among the six barriers identified by Ko and colleagues (design, selection, coordination, use, understanding, and
 874 information barriers), our work specifically addresses the *selection barrier*: the difficulty users face in knowing where to
 875 look for features and choosing appropriate tools from the available options.

The selection barrier appears in two distinct ways in block-based programming environments [10]. First, users must know that reuse mechanisms exist and where to find them within the interface. Second, even when aware of available features, users must determine when and how to apply them appropriately. Our 0% adoption rate in the standard OpenRoberta condition demonstrates that the selection barrier is difficult to overcome for domain experts without programming backgrounds, even when the interface provides the necessary functionality. Participants did not lack the capability to create custom blocks (the same individuals achieved 100% adoption in the Enhanced condition) but rather lacked the knowledge of where to look for this feature and when to apply it.

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

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

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

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

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

This aligns with theories of learning transfer and expertise development [10], suggesting that some exposure to manual processes may be valuable for teaching before introducing automation. It challenges the assumption that "easier is always better" in tool design, indicating that mental struggle during initial learning may enhance appreciation and effective use of advanced features.

937 **5.2 Implications for Practice**

938 *5.2.1 High Usability and Low Workload Support Simple Design Principles.* The SUS results (mean: 84.03) place the Reuse
 939 Assistant in the "Excellent" category, with 94% of participants (17 out of 18) rating it above the industry average of 68.
 940 This high usability score demonstrates that automated guidance can be both powerful and easy to use. The feature
 941 achieved this by focusing on simplicity: visual highlighting to show duplicates and one-click acceptance to create
 942 reusable blocks. This suggests that effective end-user features should prioritize clarity over complexity.
 943

944 The NASA-TLX workload results (mean: 2) further support this finding, showing that effective guidance does not
 945 require complex interactions. The combination of high SUS scores and low NASA-TLX workload scores demonstrates
 946 that the Reuse Assistant successfully reduces barriers without adding cognitive burden. The key to this success is
 947 directly showing users duplicate code patterns through visual highlighting and providing one-click refactoring, rather
 948 than adding complexity.
 949

950 The bimodal distribution in both SUS scores and NASA-TLX workload (with one outlier for each) suggests that while
 951 most users experience minimal burden, a small subset encounters significant difficulties. This pattern indicates individual
 952 differences in openness to automated guidance, potentially related to prior mental models, learning preferences, or
 953 comfort with system-initiated interactions.
 954

955 *5.2.2 From Passive Toolboxes to Active Assistants.* Current block-based programming environments (Scratch, Blockly,
 956 standard OpenRoberta) follow a passive interaction model where reuse mechanisms exist as features waiting to be
 957 discovered. The 0% adoption rate in the standard condition demonstrates the limitations of this approach for the
 958 end-users, as participants created functional but non-optimal solutions using linear, repetitive code structures. The 100%
 959 adoption rate with the enhanced OpenRoberta version proves that tool designers must shift from providing capabilities
 960 to actively guiding their use.
 961

962 **5.3 Threats to Validity**

963 *5.3.1 Internal Validity.*

964 *Carryover effect.* While the within subjects design allowed within-subjects comparison, the significant carryover
 965 effect ($p < 0.001$) indicates that the sequence of conditions fundamentally altered the user experience. This carryover
 966 effect means we cannot cleanly separate the impact of the Reuse Assistant from the impact of prior experience. The
 967 417 seconds performance gap between the two groups' average time differences suggests that learning from the first
 968 condition substantially influenced performance in the second condition.
 969

970 *Mitigation:* We explicitly analyzed and reported the carryover effect as a finding rather than treating it as unwanted
 971 noise. Furthermore, the experiments were balanced, meaning half of the participants performed the task with the Reuse
 972 Assistant first, then again using the standard OpenRoberta. The other half did the reverse sequence. In this way, the
 973 overall effect on the results is minimized.
 974

975 *5.3.2 External Validity.*

976 *Convenience Sampling and Population Representation.* Participants were recruited through the researchers' professional
 977 networks, creating a convenience sample. This introduces several limitations:
 978

- 979 • **Geographic and institutional diversity:** While the study included participants from multiple countries
 980 (both local and international participants connected online), recruitment relied primarily on the researchers'
 981

989 professional networks, which may not represent the full geographic and cultural diversity of potential end-users
 990 in the domain of chemistry.
 991

- 992 • **Domain representation:** While participants came from diverse scientific backgrounds (chemistry, agronomy,
 993 biochemistry) united by laboratory coursework experience, they represent primarily academic contexts rather
 994 than industrial laboratory settings where cobot programming would be used professionally.
- 995 • **Sample size:** With 18 participants for performance evaluation, usability assessment and workload assessment,
 996 the study lacks statistical power to detect small effects or to adequately characterize rare user profiles(outliers),
 997 limiting the generalizability of findings to broader populations.
 998

999 *Implications:* Findings should be interpreted as preliminary evidence rather than final proof of effectiveness across
 1000 all end-user developer populations. Replication studies with larger, more diverse samples from multiple institutions and
 1001 countries are necessary to establish the robustness of these results.
 1002

1003 *Ecological Validity: Laboratory vs. Authentic Use.* The study was conducted in a controlled setting with researcher
 1004 guidance available, tasks completed in a single session, and no real-world consequences for errors. This differs from
 1005 authentic usage where:
 1006

- 1007 • Users work independently without expert support
- 1008 • Programming tasks span multiple sessions with interruptions
- 1009 • Errors in cobot programs could damage equipment or compromise experiments
- 1010 • Users balance programming with their primary professional responsibilities

1011 *Mitigation:* We included chemistry domain experts as participants rather than generic users, and the task was based
 1012 on actual laboratory procedures. However, long-term field studies observing the Reuse Assistant in authentic work
 1013 contexts are necessary to validate its practical impact.
 1014

1015 5.3.3 Construct Validity.

1016 *Measurement Instruments.* We used standardized instruments (SUS questionnaire, NASA-TLX questionnaire) which
 1017 have established validity in usability and cognitive workload research. However:
 1018

- 1019 • **SUS limitation:** SUS assesses subjective usability perception rather than objective usability metrics, such as
 1020 error rates or task success beyond completion time.
 1021
- 1022 • **NASA-TLX limitation:** NASA-TLX assesses subjective workload perception, which may not correlate perfectly
 1023 with objective cognitive load or learning outcomes.
 1024

1025 6 Conclusion and Future Work

1026 This study examined whether automated guidance can help end-users recognize and apply code reuse in block-based
 1027 programming. We developed the Reuse Assistant, a tool that automatically detects duplicate code sequences and guides
 1028 users to create reusable custom blocks in the OpenRoberta environment.
 1029

1030 The results showed a clear difference in reuse adoption. While no participants created reusable blocks in the standard
 1031 environment, all participants successfully created reusable blocks when help from the Reuse Assistant was available.
 1032 The feature received high usability ratings (SUS mean: 84.03) and low workload scores (NASA-TLX mean: 1.92),
 1033 demonstrating that automated guidance can be both effective and easy to use.
 1034

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

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

6.1 Future Work

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

References

- [1] Felix Adler, Gordon Fraser, Eva Gründinger, Nina Körber, Simon Labrenz, Jonas Lerchenberger, Stephan Lukasczyk, and Sebastian Schweikl. 2021. Improving Readability of Scratch Programs with Search-Based Refactoring. In *Proceedings of the IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-SEET)*. IEEE. [doi:10.1109/ICSE-Companion.2021.00105](https://doi.org/10.1109/ICSE-Companion.2021.00105)
- [2] Aaron Bangor, Philip Kortum, and James T Miller. 2009. Determining what individual SUS scores mean: Adding an adjective rating scale. *Journal of usability studies* 4, 3 (2009), 114–123.
- [3] Len Bass, Paul Clements, and Rick Kazman. 2021. *Software Architecture in Practice, 4th Edition*. Addison-Wesley Professional.
- [4] Alan F. Blackwell. 2002. First Steps in Programming: A Rationale for Attention Investment Models. In *Proceedings of the IEEE Symposia on Human-Centric Computing Languages and Environments*. IEEE Computer Society, 2–10. [doi:10.1109/HCC.2002.1046334](https://doi.org/10.1109/HCC.2002.1046334)
- [5] Alan F. Blackwell and Thomas R. G. Green. 2003. Notational Systems - The Cognitive Dimensions of Notations Framework. *HCI Models, Theories, and Frameworks: Toward a Multidisciplinary Science* (2003), 103–133.
- [6] Alexander Bock and Ulrich Frank. 2021. Low-Code Platform. *Business and Information Systems Engineering* 63 (2021). [doi:10.1007/s12599-021-00726-8](https://doi.org/10.1007/s12599-021-00726-8)
- [7] John Brooke. 1996. SUS: A 'Quick and Dirty' Usability Scale. In *Usability Evaluation in Industry*, Patrick W. Jordan, Bruce Thomas, Bernard A. Weerdmeester, and Ian L. McClelland (Eds.). Taylor & Francis, London, 189–194.
- [8] Jay L Devore. 2015. *Probability and Statistics for Engineering and the Sciences* (9th ed.). Cengage Learning, Boston, MA.
- [9] Sandra G Hart and Lowell E Staveland. 1988. Development of NASA-TLX (Task Load Index): Results of empirical and theoretical research. In *Human Mental Workload*, Peter A Hancock and Najmedin Meshkati (Eds.). North-Holland, Amsterdam, 139–183.
- [10] Andrew J. Ko, Brad A. Myers, and Htet Htet Aung. 2004. Six Learning Barriers in End-User Programming Systems. *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)* (2004), 199–206. [doi:10.1109/VLHCC.2004.47](https://doi.org/10.1109/VLHCC.2004.47)
- [11] Yuhan Lin and David Weintrop. 2021. The Landscape of Block-Based Programming: Characteristics of Block-Based Environments and How They Support the Transition to Text-Based Programming. *Journal of Computer Languages* 67 (2021), 101075. [doi:10.1016/j.cola.2021.101075](https://doi.org/10.1016/j.cola.2021.101075)
- [12] Hugo Lourenço, Carla Ferreira, and João Costa Seco. 2021. OSTRICH - A Type-Safe Template Language for Low-Code Development. In *2021 ACM/IEEE 24th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. 216–226. [doi:10.1109/MODELS50736.2021.00030](https://doi.org/10.1109/MODELS50736.2021.00030)
- [13] Vlad Magdalin. 2012. Low code platform tool Webflow. <https://webflow.com/>.
- [14] Derek Roos. 2005. Low code platform tool Mendix. <https://www.mendix.com/>.
- [15] Bernard L Welch. 1947. The generalization of 'Student's' problem when several different population variances are involved. *Biometrika* 34, 1-2 (1947), 28–35. [doi:10.1093/biomet/34.1-2.28](https://doi.org/10.1093/biomet/34.1-2.28)
- [16] Roel J. Wieringa. 2014. *Design Science Methodology for Information Systems and Software Engineering*. Springer, Berlin, Heidelberg. [doi:10.1007/978-3-662-43839-8](https://doi.org/10.1007/978-3-662-43839-8)

A System Usability Scale (SUS) Questionnaire

The System Usability Scale (SUS) is a widely used standardized questionnaire for assessing the perceived usability of a system. Participants respond to each statement using a 5-point Likert scale ranging from 1 (Strongly Disagree) to

¹⁰⁹³ 5 (Strongly Agree). The SUS score is calculated by converting the responses to a scale of 0-100, where higher scores
¹⁰⁹⁴ indicate better usability.
¹⁰⁹⁵

¹⁰⁹⁶ A.1 SUS Statements

- ¹⁰⁹⁸ (1) I think that I would like to use the Reuse Assistant feature frequently.
- ¹⁰⁹⁹ (2) I found the Reuse Assistant feature unnecessarily complex.
- ¹¹⁰⁰ (3) I thought the Reuse Assistant feature was easy to use.
- ¹¹⁰² (4) I think that I would need the support of a technical person to be able to use the Reuse Assistant feature.
- ¹¹⁰³ (5) I found the various functions in the Reuse Assistant feature were well integrated.
- ¹¹⁰⁴ (6) I thought there was too much inconsistency in the Reuse Assistant feature.
- ¹¹⁰⁵ (7) I would imagine that most people would learn to use the Reuse Assistant feature very quickly.
- ¹¹⁰⁶ (8) I found the Reuse Assistant feature very cumbersome to use.
- ¹¹⁰⁸ (9) I felt very confident using the Reuse Assistant feature.
- ¹¹⁰⁹ (10) I needed to learn a lot of things before I could get going with the Reuse Assistant feature.

¹¹¹² A.2 Scoring Method

¹¹¹⁴ We calculate the System Usability Scale (SUS) score based on the standard method defined by Brooke [7]. The process
¹¹¹⁵ consists of three steps:

- ¹¹¹⁷ (1) **Normalize the responses:**
 - ¹¹¹⁸ • For **odd-numbered** items (1, 3, 5, 7, 9), subtract 1 from the user response.
 - ¹¹¹⁹ • For **even-numbered** items (2, 4, 6, 8, 10), subtract the user response from 5.
- ¹¹²¹ This converts every answer to a range of 0 to 4.
- ¹¹²² (2) **Sum the contributions:** Add the normalized scores for all 10 items together.
- ¹¹²³ (3) **Scale the total:** Multiply the sum by 2.5 to convert the range from 0–40 to 0–100.

¹¹²⁵ Formally, if x_i is the score for the i -th question, the total SUS score is given by:

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

¹¹³⁰ B NASA-TLX post-task Questionnaire

¹¹³² The NASA Task Load Index (NASA-TLX) [9] is a multi-dimensional questionnaire that provides an overall workload
¹¹³³ score based on a weighted average of ratings on six subscales. For this study, we utilized the "Raw TLX" method
¹¹³⁴ (unweighted). Participants rated the following six dimensions on a scale from 1 (Very Low) to 10 (Very High), which
¹¹³⁵ were then normalized to a 0–100 scale.

- ¹¹³⁸ (1) **Mental Demand:** How much mental and perceptual activity was required (e.g., thinking, deciding, calculating,
¹¹³⁹ remembering, looking, searching, etc.)? Was the task easy or demanding, simple or complex, exacting or
¹¹⁴⁰ forgiving?
- ¹¹⁴¹ (2) **Physical Demand:** How much physical activity was required (e.g., pushing, pulling, turning, controlling,
¹¹⁴² activating, etc.)? Was the task easy or demanding, slow or brisk, slack or strenuous, restful or laborious?

- 1145 (3) **Temporal Demand:** How much time pressure did you feel due to the rate or pace at which the tasks or task
1146 elements occurred? Was the pace slow and leisurely or rapid and frantic?
1147
1148 (4) **Performance:** How successful do you think you were in accomplishing the goals of the task set by the
1149 experimenter (or yourself)? How satisfied were you with your performance in accomplishing these goals?
1150
1151 (5) **Effort:** How hard did you have to work (mentally and physically) to accomplish your level of performance?
1152
1153 (6) **Frustration Level:** How insecure, discouraged, irritated, stressed and annoyed versus secure, gratified, content,
relaxed and complacent did you feel during the task?

1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196