

1 Title 3

2 ANNE-MARIE ROMMERDAHL, SDU, Denmark

3 JEREMY ALEXANDER RAMÍREZ GALEOTTI, SDU, Denmark

4 DIMITRIOS DAFNIS, SDU, Denmark

5 NASIFA AKTER, SDU, Denmark

6 MOHAMMAD HOSEIN KARDOUNI, SDU, Denmark

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

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

9 ACM Reference Format:

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

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

12 ACM, New York, NY, USA, 16 pages. <https://doi.org/XXXXXXX.XXXXXXX>

13 1 Introduction

14 2 Background and Related Work

15 Software reuse is a broad term, that refers to the practice of reusing previously written code, rather than coding from scratch. It is such an important part of software engineering, that one of the ways to measure the quality of software is by its 'Reusability'[2] - i.e. the degree to which the application or its components can be reused. There are multiple benefits to practicing reuse in software engineering. One developer could save time by using another developer's reusable component, rather than coding their own. The developer avoids both the work of writing the syntax and designing the logic of the component. The developer can design their own reusable components, keeping all the logic in one place, which can then be tested thoroughly. However, despite reuse being an important practice in software engineering, there is still a limited focus on this practice when it comes to low-code development platforms (LCDP).

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

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

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

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

20 Manuscript submitted to ACM

21 Manuscript submitted to ACM

53 There have been proposed some ideas on how to promote reuse for LCPs, such as the strongly-typed rich templating
 54 language OSTRICH, developed for the model-driven low-code platform OutSystems[6]. OutSystems provides scaffolding
 55 mechanisms for common development patterns and sample screen templates, both designed by experts on domain-
 56 specific languages (DSL). The practice of using templates in the OutSystems platform involves cloning and modifying
 57 samples, which may require more knowledge than the end-user possesses. The goal of OSTRICH is to remove this need
 58 for adaptation when using templates, to remove the knowledge-barrier when making use of the available templates. This
 59 is done by abstracting and parameterizing the templates. A limitation of OSTRICH, is that it currently only supports
 60 the top nine most used production-ready screen templates from OutSystems. The end-user may not create and save
 61 their own templates, nor can they re-apply a template which they have customized.
 62

63 Another approach focused on enabling reuse of models, by converting and merging models into a single graph (the
 64 Knowledge Graph), which acts as a repository of models[4]. This graph is used to provide recommendations to the
 65 end-user, based on the model they're currently building. While this feature of recommending models (either constructed
 66 by domain experts and then developed by model experts, or made by the end-user themselves) could prove very useful,
 67 the study is clearly not focused on guiding the user towards reusing their own models.
 68

69 Building on the ideas discussed for improving reuse in low-code development platforms (LCDPs), several popular
 70 tools show these concepts in action. For instance, Webflow[7] is a leading low-code platform that offers a wealth of
 71 features for building responsive websites. One of its standout features is the ability to create reusable components and
 72 UI kits, which can significantly speed up the development process. With Webflow's intuitive interface, developers can
 73 quickly design and prototype components, and then reuse them across multiple pages and projects. Despite all of the
 74 useful features that this tool has, it does not provide guidance to the end-users to create custom reusable components.
 75

76 In a similar way, Mendix[8] takes this further for full enterprise apps by offering shareable building blocks like
 77 simple actions (microflows) and UI parts that anyone on a team can grab and use again without recoding. Through its
 78 Marketplace, a free online hub, you can download ready templates, connectors for tools like Salesforce, and basic setups
 79 that fit right into new projects, making everything faster and more uniform. This approach builds on the flexibility seen
 80 in platforms like Webflow, but adds strong team tools and AI suggestions to spot and create reusable pieces, empowering
 81 even beginners to build complex apps while keeping reuse simple and widespread. This tool does offer guidance for the
 82 end-users to create custom reusable components through its AI suggestions, a lot of times these suggestions are not
 83 accurate enough (how do we know this??*).
 84

85 OutSystems[9] further enhances the concept of reuse in low-code development platforms by emphasizing rapid
 86 application delivery through its robust set of features. Like Webflow and Mendix, OutSystems also provides a library of
 87 reusable components and templates that help developers complete projects faster. Its user-friendly visual development
 88 environment allows users to easily drag and drop elements while connecting with existing systems. OutSystems also
 89 supports teamwork with built-in version control and feedback features, making it easy for teams to share and improve
 90 reusable components. Additionally, the platform uses AI to suggest the best solutions and components for specific
 91 tasks. By encouraging reuse at both individual and team levels, OutSystems enables organizations to create scalable
 92 applications quickly while ensuring quality and consistency. Similarly to the previous tool explained, the AI suggestions
 93 that this tool provides are not always accurate to successfully guide the end-user to create custom reusable components
 94 (again, how do we know this??*).
 95

96 In order to analyze how block-based robotics environments address reuse area, 4 representative platforms were
 97 compared: mBlock, MakeCode, SPIKE LEGO, VEXcode GO and Open Roberta. The comparison focused on three main
 98

105 dimensions of reuse: structural reuse (through user-defined blocks or functions), social reuse (through sharing or
 106 remixing existing projects), and interoperable reuse (through import/export capabilities).
 107

108
 109 Table 1. Block Based Robotics Environments Reuse Support
 110
 111

| 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 |

112
 113 In this context, “reuse support” represents a scale that measures how effectively each platform facilitates reuse-related
 114 features. High reuse support indicates that users can easily create, share, and adapt existing components or projects.
 115 Medium reuse support suggests that some reuse mechanisms are available but limited in scope or flexibility. Low reuse
 116 support implies that the platform provides only minimal or restricted features to promote reuse and improve user
 117 productivity.
 118

119 As shown in Table 1, although these platforms include reusability features, they are quite limited, as none of them
 120 provide users with clear guidance on how to use these tools effectively, which restricts their ability to fully leverage
 121 them.
 122

123 Lin and Weintrop (2021) noted that most existing research on block-based programming focuses on supporting
 124 the transition to text-based languages rather than exploring how features within BBP environments [5]—such as
 125 abstraction or reuse—can enhance learning outcomes. In contrast, our work emphasizes guided abstraction, helping
 126 users understand and practice modular design directly within block-based environments.
 127

128 Techapalokul and Tilevich (2019) proposed extending the Scratch programming environment with facilities for
 129 reusing individual custom blocks to promote procedural abstraction and improve code quality. They observed that
 130 while Scratch enables remixing of entire projects, it lacks mechanisms for reusing smaller, modular pieces of code. Their
 131 work suggests that supporting such fine-grained code reuse could enhance programmer productivity, creativity, and
 132 learning outcomes. Building on this idea, our project applies similar principles within the OpenRoberta environment
 133 by automating the detection of duplicate code segments and guiding users toward creating reusable custom blocks.
 134 Adler et al. (2021) introduced a search-based refactoring approach to improve the readability of Scratch programs by
 135 automatically applying small code transformations, such as simplifying control structures and splitting long scripts.
 136 Their findings demonstrated that automated refactoring can significantly enhance code quality and readability for
 137 novice programmers. Building upon this concept, our project applies similar principles in the OpenRoberta environment,
 138 focusing on detecting duplicate code segments and guiding users toward creating reusable custom blocks to promote
 139 modularity and abstraction.[1].
 140

141 Existing block-based environments provide mechanisms for reuse, but lack intelligent support to help users recognize
 142 and apply reuse in practice. To address this gap, our project introduces a guided reuse assistant within the Open Roberta
 143 Lab environment. The tool is designed to help users identify and apply reuse more easily while creating their robot
 144 programs. It works by automatically scanning a user’s block-based program to detect repeated code segments in the
 145

¹⁵⁷ workspace. The system visually highlights the found duplicates, drawing the user's attention to patterns that could be
¹⁵⁸ simplified.

¹⁵⁹ The tool also offers the functionality to create the custom block for the end-user, by identifying the small differences
¹⁶⁰ between the repeated parts—such as numbers, variables, or parameters—and turning these differences into inputs for
¹⁶¹ the new block. The tool automatically replaces all relevant duplicate sequences with the new custom block.

¹⁶² By combining ideas from procedural abstraction (organizing code into meaningful, reusable parts) and automated
¹⁶³ refactoring (improving code through intelligent transformations), our tool aims to make block-based programming
¹⁶⁴ more structured and efficient. It encourages users to build programs that are modular and easier to maintain, helps
¹⁶⁵ reduce unnecessary repetition, and supports learning by making the concept of reuse clear and hands-on.

¹⁷⁰ 3 Study Design

¹⁷¹ Following the Design Science methodology, our study is structured into three main phases: problem investigation to
¹⁷² define goals, treatment design to specify the artifact requirements, and treatment validation to assess the artifact's
¹⁷³ performance in a controlled environment.

¹⁷⁸ 3.1 Problem Investigation

¹⁷⁹ *3.1.1 Problem Context and Motivation.* End-user development (EUD) for collaborative robots (cobots) presents unique
¹⁸⁰ challenges, particularly for users without formal programming training. In domains such as chemistry laboratories,
¹⁸¹ educational robotics, and industrial settings, end-users need to program robots to perform specific tasks but often lack
¹⁸² the software engineering knowledge to write maintainable, well-structured code. In the domain of Chemistry, one of
¹⁸³ the most relevant and important tasks is performing experiments in labs in order to test a hypothesis, or to aid in the
¹⁸⁴ understanding of how chemicals react. Robots can be used in chemistry labs to automate experiments with great effect,
¹⁸⁵ as many experiments involve steps that are repetitive, and susceptible to human error, such as a step being overlooked,
¹⁸⁶ instructions being misread, etc. Automation of menial tasks will leave the chemists with more time for other work,
¹⁸⁷ and also comes with the added bonus of chemists not having to handle dangerous chemicals. One critical challenge in
¹⁸⁸ EUD is code reuse. Users frequently create repetitive code because they struggle to recognize duplicate patterns, lack
¹⁸⁹ knowledge about abstraction mechanisms, or find existing tools too complex to use effectively. This problem manifests
¹⁹⁰ in several ways: programs become unnecessarily long and difficult to maintain and small changes require modifications
¹⁹¹ in multiple locations, increasing the risk of errors. Several visual programming environments, like OpenRoberta Lab,
¹⁹² don't provide assistance in identifying when code should be reused or how to extract repeated sequences into reusable
¹⁹³ components. As lab work in chemistry involves many repetitive tasks, these challenges can easily become an obstacle
¹⁹⁴ for the chemists, which may turn them away from using cobots, as the inconvenience outweighs the benefits.

¹⁹⁵ *3.1.2 Stakeholder Analysis.* Chemists and lab technicians who use cobots for repetitive tasks such as sample prepa-
¹⁹⁶ ration, dispensing, mixing, and quality control procedures. They possess deep domain expertise in chemistry but
¹⁹⁷ limited programming knowledge, often creating long, repetitive programs that become difficult to maintain when
¹⁹⁸ adapting experimental protocols. Their primary need is to quickly create and modify robot programs without becoming
¹⁹⁹ programming experts.

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
213 the user's workspace. These requirements are necessary to help the end-user recognize opportunities for reuse, that
214 would otherwise go unnoticed. Once detected, the system must suggest the creation of reusable custom blocks, allowing
215 the user to accept or reject these suggestions. These signals are important, as they give the end-user control over the
216 reuse process, allowing them to decide when and how to apply reuse in their programs. Regarding non-functional
217 requirements, the artifact must seamlessly integrate with the existing Open Roberta Lab environment to ensure a
218 smooth user experience. The interface should be intuitive for end-users, minimizing the learning curve and making it
219 easy to understand and use the reuse features. Additionally, the artifact should not interfere with the existing workflow,
220 allowing users to continue their programming tasks without disruption. Finally, clear visual feedback during the
221 detection process is essential to help users understand what the system is doing and how to respond to its suggestions.
222

223 3.2.1 *Artifact Specification: The Reuse Assistant.* To satisfy the requirements above, we designed the Reuse Assistant as
224 an extension of Open Roberta Lab.
225

226 3.2.2 *Architecture.* The system enables the execution of block-based programs on a simulated cobot through a three-tier
227 architecture, as illustrated in 1. The workflow consists of the following stages:
228

- 229 (1) **Client Side (Open Roberta):** The user interacts with the Open Roberta UI to assemble block sequences. The
230 Reuse Assistant operates at this layer, analyzing blocks in real-time. Upon execution, the client generates specific
231 data structures ("Generated Headers") representing the program logic.
232
- 233 (2) **Backend (Flask Server):** The client transmits these headers via HTTP POST requests to a Flask-based API
234 Endpoint. A "Translator" component processes the data, mapping the abstract block definitions to concrete
235 Python methods compatible with the robot's control logic.
236
- 237 (3) **Simulation (Mujoco):** The mapped methods trigger the execution of commands within the Mujoco Simulator,
238 which renders the physical behavior of the cobot in the virtual environment.
239

240 3.2.3 *Detection Algorithm.* exttthighlightOnlyFunctionCandidates function. The approach is intentionally simple so it
241 is easy to read and to implement in a real block editor. The algorithm follows three main steps:
242

243 highlightOnlyFunctionCandidates function. The algorithm operates in several steps:
244

- 245 • **Linearization:** First, the algorithm linearizes the block workspace into a sequential list of blocks.
246
- 247 • **Identify sequences:** It then iterates through this list to identify all possible sequences of blocks that meet a
248 minimum unique block type length requirement (three blocks) that can be repeated more than once.
249
- 250 • **Sequences Matching:** If the same sequence of block types is found more than once, it will be added to the
251 CustomReusableCandidates list which will eventually be sorted by longest and most recent duplicated sequences.
252 In the end the highest priority candidate gets returned.
253

254 The pseudocode below is short, explicit, and uses straightforward data structures (lists).
255

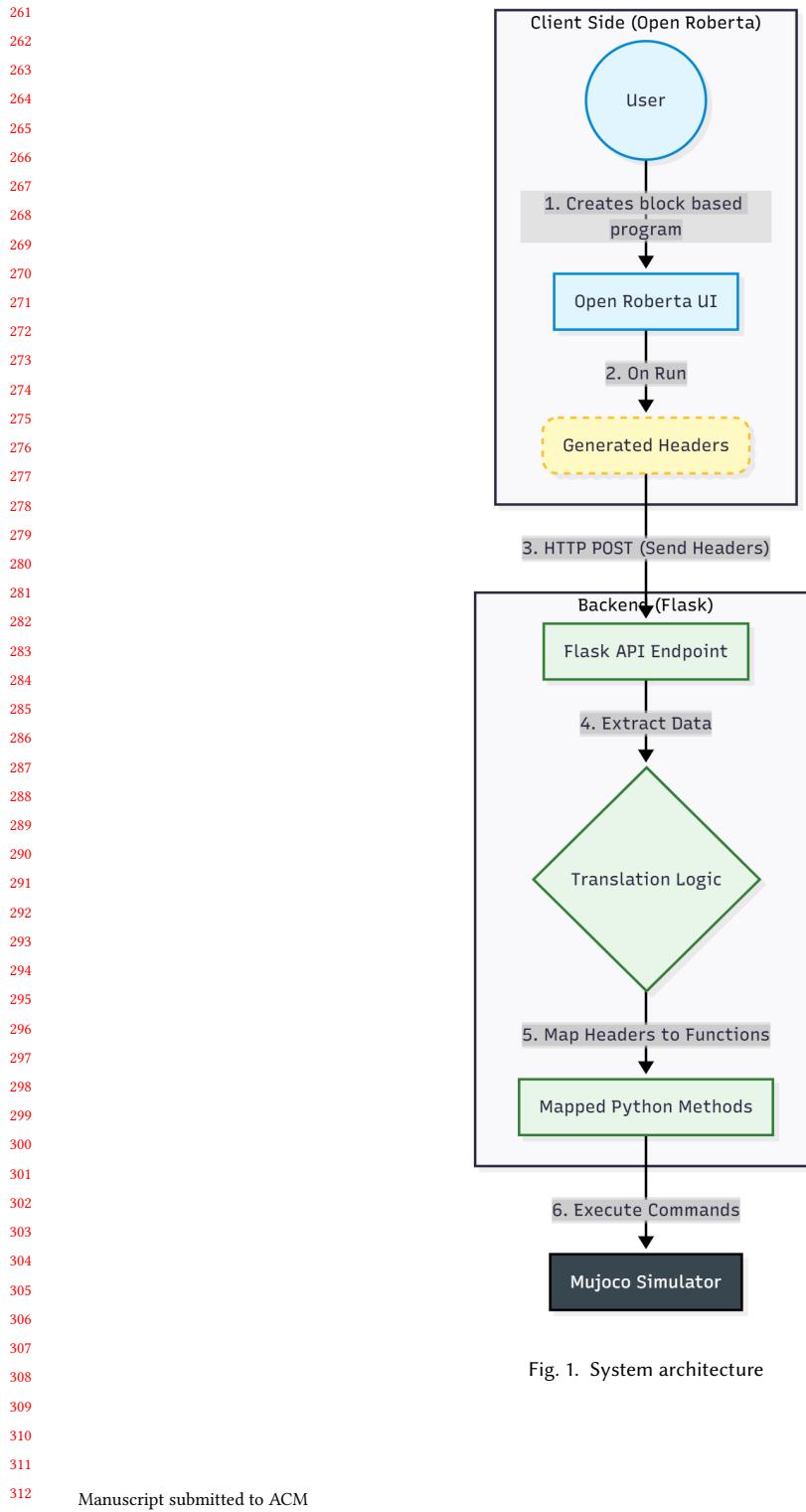


Fig. 1. System architecture

Algorithm 1 Duplicate Sequence Detection

```

313
314 Require: Workspace, StartBlock // user's block workspace
315
316 Require: MinimumSequenceLength = 3, MinimumDifferentBlockTypesInSequence = 3, MaxSequenceLength = 10
317 Ensure: ReusableComponentCandidates // list of repeated block sequences to return
318 1: Chain = buildLinearChain(StartBlock)
319 2: Sequences = List<sequence>
320 3: for startIndex = 0 to length(Chain) - 1 do
321 4:   for sequenceLength = 1 to MaxSequenceLength do
322 5:     sequence = Chain[startIndex .. startIndex + sequenceLength - 1]
323 6:     numberofBlockTypesInSequence = getNumberOfDistinctBlockTypes(sequence)
324 7:     if sequenceLength >= MinimumSequenceLength and numberofBlockTypesInSequence >= MinimumDiffer-
325     entBlockTypesInSequence then
326 8:       Sequences.append(sequence) // record sequence occurrence
327 9:     end if
328 10:   end for
329 11: end for
330 12: ReusableComponentCandidates = {Sequences | occurrence ≥ 2}
331 13: sort ReusableComponentCandidates by (longest sequence length and most recent occurrence)
332 14: return ReusableComponentCandidates[0] // Return highest priority candidate
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364

```

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

3.2.4 *User Interface and Interaction.* The user interface is designed to be intuitive and non-disruptive. When the detection algorithm identifies a candidate, the system visually highlights the blocks on the canvas as illustrated in Figure 2. A non-blocking toast notification appears, prompting the user to confirm the refactoring. If confirmed, the system automatically generates the custom block definition in a dedicated workspace area (handling visibility via `revealDefinitionWorkspacePane`) and updates the main workspace, replacing the redundant code with concise function calls as shown in Figure 3. This process abstracts the complexity of manual function creation, guiding the user toward modular design practices.

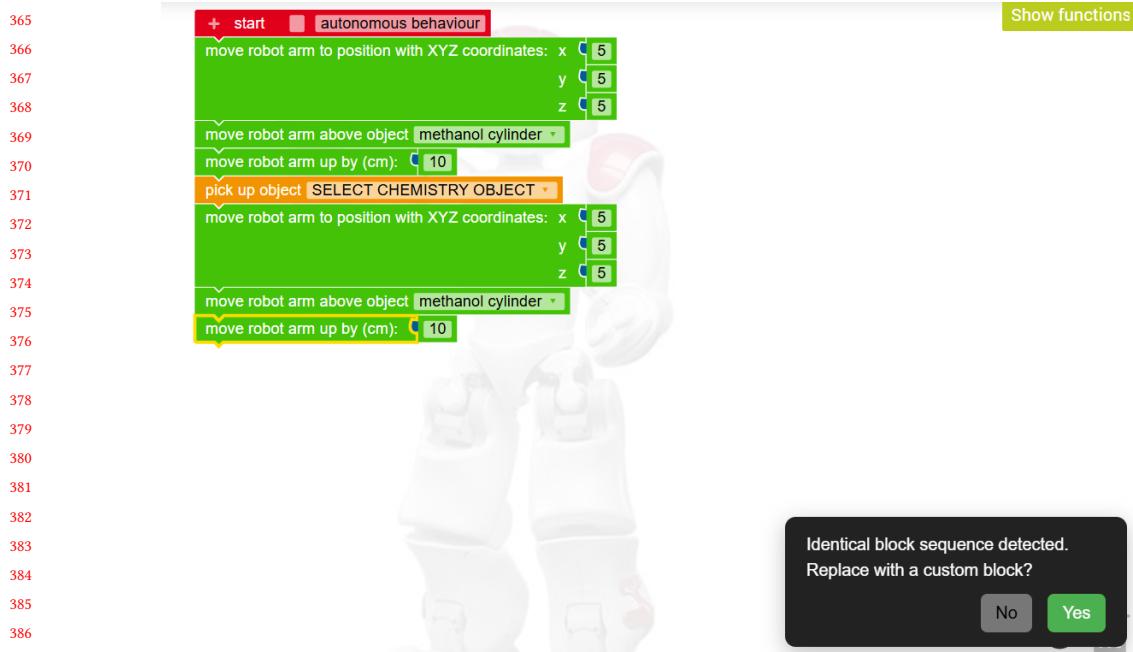


Fig. 2. Reuse Assistant workflow — detection: the interface detects and highlights duplicate blocks by changing their color to green.

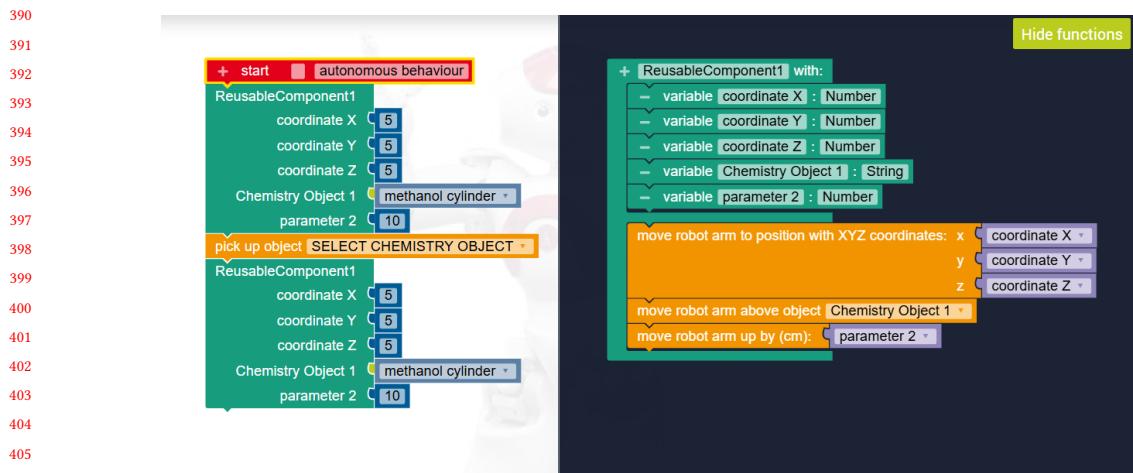


Fig. 3. Reuse Assistant workflow — refactoring: the automated refactoring result, showing the new custom block definition and the simplified main program.

3.3 Treatment Validation

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

Manuscript submitted to ACM

417 3.3.1 *Participant Recruitment.* A total of 10 participants will be selected to ensure a diverse range of experience levels
418 with block-based programming. Time constraints and resource availability have influenced the decision to limit the
419 number of participants. Participants will be recruited from a diverse pool of individuals affiliated with the University
420 of Southern Denmark and the broader chemistry community. This group of participants includes chemistry teachers,
421 professional chemical engineers, and students currently enrolled in chemistry-intensive curricula. To ensure relevant
422 practical expertise, the selection specifically targets those who frequently engage in laboratory environments. The
423 experimental sessions will be conducted across a range of environments to accommodate participant availability.
424 Physical sessions will take place within the chemistry laboratories at the University of Southern Denmark (SDU) as
425 well as a private residential setting. For remote participants, sessions will be administered virtually using Discord for
426 communication and AnyDesk for remote desktop control.
427

428 3.3.2 *Ethical Considerations and Sampling.* Prior to the commencement of the study, all participants are required to sign a
429 consent form acknowledging their voluntary participation and granting permission for screen recording and data usage.
430 It should be noted that this recruitment strategy constitutes *convenience sampling*. As such, they may not represent the
431 general population.
432

433 3.3.2 *Task Execution.* The participants will initially be given a short introduction to the OpenRoberta UI, as well
434 as the mujoco robot simulator. They will then perform one task which is described by a set of pre-defined steps to
435 perform. This task has been specifically designed to promote the reusability aspect. The task is focused on the domain
436 of chemistry, as it is modelled after a real lab experiment performed by chemistry students at SDU.
437

438 The participants will be instructed to program the robot to execute the following sequence of operations:
439

- 440 (1) Move the robot arm above mix cylinder
- 441 (2) Mix the chemistry ingredients
- 442 (3) Move the robot arm above the analysis pad
- 443 (4) Analyze the sample
- 444 (5) If the solution is analyzed (use if statement) then show a response message in the laptop's screen
- 445 (6) Place the following three objects into their corresponding slots in the chemistry equipment toolbox:

- 446 • Methanol cylinder
- 447 • Chloroform syringe
- 448 • Toluene syringe

- 449 (7) Important notes for the participants:

- 450 • After placing an object to its slot in the toolbox **wait 2 seconds** before you move to pick a new one.
- 451 • After placing the **chloroform syringe** to its slot, **move the robot arm up by 10 cm** before you move to pick
452 the next chemistry object
- 453 • Click the **play** button on the bottom right corner to start the simulation
- 454 • Click the **reset** button on the bottom right corner to reset the scene of the robot simulator

455 Most optimal solution pre-defined by the researchers:
456

457 Instead of creating a long linear sequence of blocks (hard-coding the movement for all three objects), the most
458 optimal solution utilizes a **“Custom Reusable Component”** to handle the repetitive action of placing an object to its
459 corresponding slot inside the equipment toolbox. This approach not only reduces redundancy but also enhances code
460 maintainability and readability, aligning with best practices in software development.
461

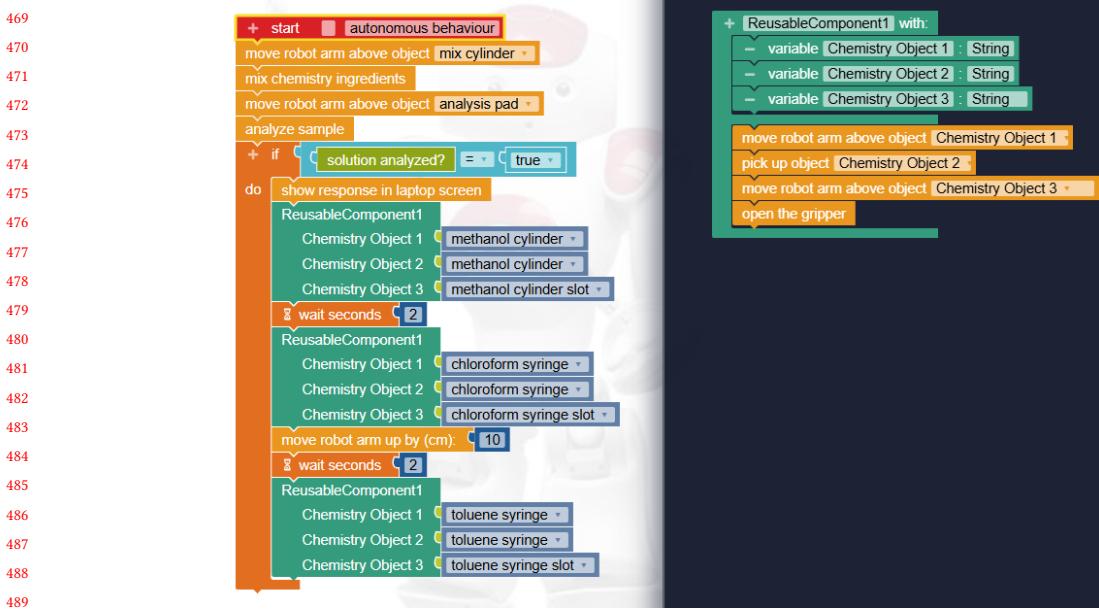


Fig. 4. The optimal solution implemented in OpenRoberta, utilizing a custom block for the object placement sequence.

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:** Comparing the participants who will first use the enhanced version of OpenRoberta against those who will first use the standard version.
- (2) **Solution Accuracy:** Evaluated by comparing the participant's block configuration against the pre-defined optimal solution.
- (3) **Survey Feedback:** Collected via a post-experiment survey designed to capture demographic data and subjective perceptions of the utility of the block creation guidance features.

This comprehensive evaluation will provide a detailed understanding of how useful and effective is the block creation guidance feature to the end-users.

4 Results

The treatment validation was concluded with a total of 10 participants. The analysis of the collected data combines quantitative metrics regarding user preference and satisfaction with qualitative feedback derived from survey responses.

Manuscript submitted to ACM

521 **4.1 Performance Evaluation**

522
523 To evaluate the efficiency and effectiveness of the proposed reusable component features, we analyzed two primary
524 metrics: Task Completion Time and Solution Accuracy.

525
526 *4.1.1 Task Completion Time.* The total time required to complete the experimental task was recorded for both the
527 *Standard* and *Enhanced* conditions.

528 We compared the performance of participants based on the order of conditions (see Table 2). The analysis reveals a
529 significant reduction in task duration when using the Enhanced version. The average completion time for the participants
530 that used the Enhanced version first was 8.5 minutes, compared to 10 minutes for the Standard version.

$$532 \quad \text{Efficiency Improvement} = \frac{10.0 - 8.5}{10.0} \times 100\% = 15\% \quad (1)$$

533
534

535
536 Table 2. Breakdown of Mean Task Completion Times
537

| 538 Experimental Condition | 539 Mean Time (min) |
|---|----------------------------|
| 540 <i>Group of Participants that used the Enhanced OpenRoberta Version First</i> | 8.5 |
| 541 <i>Group of Participants that used the Standard OpenRoberta Version First</i> | 10.0 |

542
543 *4.1.2 Solution Accuracy.* Solution accuracy was evaluated by comparing participant solutions against the optimal
544 reference solution defined in the treatment evaluation.

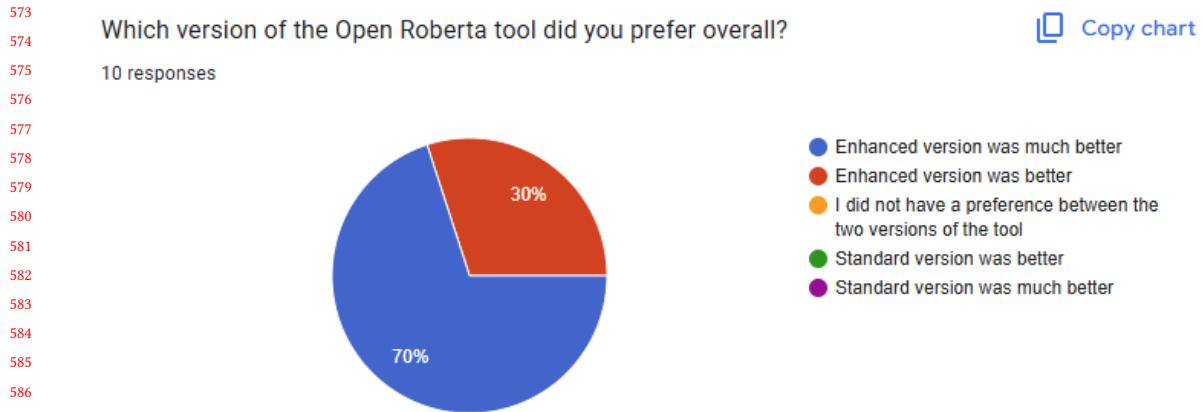
545 *Adoption of Reusable Blocks.* A key metric was the voluntary adoption of the custom reusable component. In the
546 *Enhanced* version, 10/10 participants successfully implemented a custom reusable block to handle the repetitive object
547 placement steps. In contrast, in the *Standard* condition, participants predominantly relied on linear, repetitive code
548 structures. Without the guidance features, none of them recognized the opportunity to create a reusable block.

549 **4.2 Survey Quantitative Results**

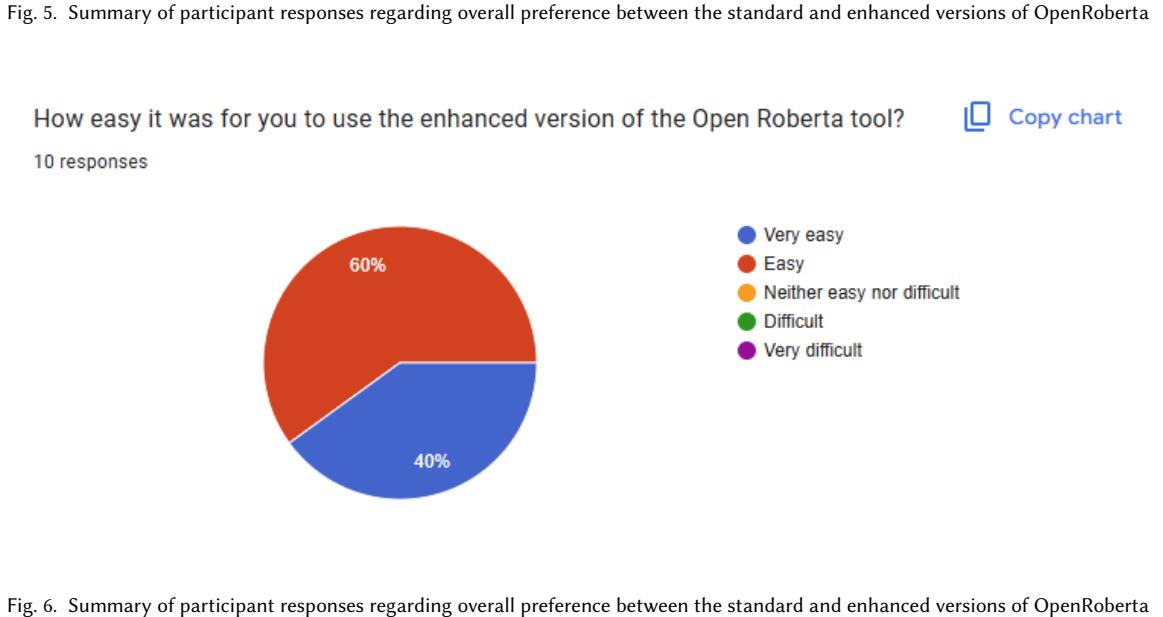
550 *4.2.1 User preference between Standard and Enhanced Versions of OpenRoberta.* The survey results indicate a unanimous
551 preference for the enhanced version of the OpenRoberta Lab. As illustrated in Figure 5, 70% of participants rated the
552 enhanced version as “much better” than the standard version, while the remaining 30% rated it as “better.” No participants
553 preferred the standard version or rated the two versions as equivalent.

554
555 *4.2.2 Usability of the Guidance Feature.* Regarding usability of the enhanced OpenRoberta version, we received high
556 acceptance scores. As illustrated in Figure 6, 40% of participants found the enhanced version “very easy” to use, and
557 60% rated it as “easy.” No participants rated the enhanced version as “Neither easy nor difficult,” “Difficult,” or “Very
558 difficult” to use.

559
560 *4.2.3 Evaluation of the Visual Highlighting.* A key component of the enhanced version was the visual highlighting
561 designed to guide the user into an automatic custom reusable block creation. As shown in Figure 7, results showed a
562 high level of user satisfaction, with 90% of participants reporting they were either “satisfied” (20%) or “very satisfied”
563 (70%) with the features. Only one participant (10%) expressed a neutral stance.



[Copy chart](#)

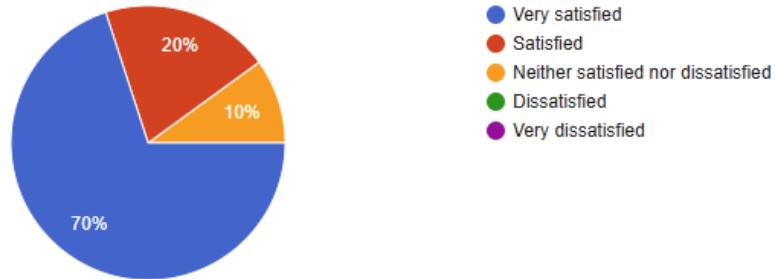


4.3 Qualitative Feedback

The post-experiment survey included open-ended questions to gather detailed feedback. The thematic analysis of these responses revealed two primary findings:

Manuscript submitted to ACM

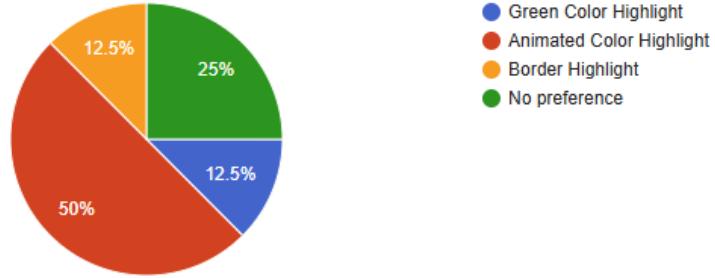
625 How satisfied were you with the visual highlight?
 626
 627 10 responses
 628
 629
 630
 631
 632
 633
 634
 635
 636
 637
 638
 639
 640
 641



[Copy chart](#)

Fig. 7. Summary of participant responses regarding overall preference between the standard and enhanced versions of OpenRoberta

642
 643 Which highlight option did you prefer the most?
 644
 645 8 responses
 646
 647
 648
 649
 650
 651
 652
 653
 654
 655
 656
 657
 658
 659
 660
 661
 662



[Copy chart](#)

Fig. 8. Summary of participant responses regarding overall preference between the standard and enhanced versions of OpenRoberta

663 *Efficiency and Speed.* When asked to identify the biggest difference between the two versions, the majority of participants cited *efficiency*. Responses frequently described the enhanced version as “faster” and noted that it “saved a lot of time.” This aligns with the quantitative preference data, suggesting that the usability features successfully reduced the perceived workload.

664
 665
 666
 667
 668
 669
 670
 671 *Suggestions for Improvement.* Participants also provided constructive feedback regarding the function blocks. Two participants specifically suggested that the system should more clearly “*specify parameter names*” within the function blocks to improve clarity. Another participant noted that the function call block should be pre-configured for immediate use in the blockchain. These suggestions highlight a need for clearer labeling in future iterations of the interface.

677 5 Discussion

678 5.1 Lessons Learned

680 Utilizing OpenRoberta Lab as a representative block-based robotics environment, this study examined the efficacy of
 681 automated guidance mechanisms in promoting software reuse among chemistry students and educators engaged in
 682 laboratory experimentation.
 683

684 Based on the feedback from the participants, as well as observations of how they solved the task, the participants
 685 found the enhanced version of OpenRoberta Lab to be better than the standard version. Noteably, 9 out of 10 participants
 686 commented on how the enhanced version let them perform their task faster. As described in section 2, this is also one
 687 of the main benefits of reuse in the field of software engineering.
 688

689
 690 **5.1.1 Overcoming the Recognition Barrier for Reuse.** A defining finding of this study is the contrast in adoption rates:
 691 100% of participants utilized reusable blocks in the *Enhanced* version of OpenRoberta Lab, compared to 0% in the
 692 *Standard*. This confirms the literature cited in Section 1 regarding the high barrier to entry for "Citizen Developers".
 693 Despite the task being repetitive by design, participants in the standard environment prioritized immediate task
 694 completion over code optimization (linear programming). The *Enhanced* version successfully shifted this behavior not
 695 by forcing reuse, but by lowering the cognitive cost of identifying opportunities. This suggests that for domain experts
 696 like chemists, the barrier to reuse is not a lack of utility, but a lack of recognition.
 697
 698

699
 700 **5.1.2 Impact of Automated Construction of Reusable Components.** The 15% reduction in task completion time highlights
 701 the value of automating the block creation process. In the standard environment, creating a reusable component requires
 702 a manual, multi-step process of defining a function and relocating blocks. The enhanced version streamlined this by
 703 automating the structural setup of the custom block once a duplicate was detected. This confirms that removing the
 704 "friction" of manual block assembly is crucial for encouraging reusability among non-programmers.
 705
 706

707 **5.1.3 Visual Salience in Learning.** The user preference for the *Animated Color Highlight* (50% preference) and the
 708 high satisfaction rates (90% satisfied/very satisfied) underscore the importance of visual salience. In a dense visual
 709 environment like OpenRoberta, static cues are easily overlooked. The dynamic nature of the animation acted as a
 710 "Just-in-Time" trigger, interrupting the user's tunnel vision exactly when the redundancy occurred. This supports the
 711 use of proactive, visually distinct interruptions in educational IDEs to correct inefficient patterns in real-time.
 712
 713

714 **5.1.4 Suggestions by Participants.** Changes suggested by the participants mainly focus on smaller customizations of
 715 the tool and the OpenRoberta Lab UI. It would be amiss to claim that the lack of suggested changes, focused on the tool
 716 overall, indicate that there is no need for improvement of the tool. As many of the participants consider themselves
 717 'beginners' in regards to Computer Programming, it's likely that they lack ideas about other ways the tool could have
 718 been designed. Instead, these answers can be interpreted as the participants having little to no issue with the current
 719 design.
 720
 721

722 5.2 Implications for Practice

723 The findings of this study have broader implications for the design of End-User Development (EUD) environments and
 724 educational technology. The success of the enhanced OpenRoberta interface suggests three key shifts for future tool
 725 development:
 726
 727

729 5.2.1 *Transitioning from Passive to Proactive Environments.* Current block-based environments (such as Scratch or
730 standard OpenRoberta) largely rely on a *passive* interaction model, where advanced features like "Functions" sit in
731 a toolbox waiting to be discovered. Our study demonstrates that domain experts (e.g., chemists) often fail to utilize
732 these features voluntarily, even when they would be beneficial. The 100% adoption rate in the Enhanced condition
733 implies that EUD tools must evolve into *active assistants*. Development environments should incorporate background
734 monitoring systems that detect inefficient patterns (such as code duplication) and proactively intervene with architectural
735 suggestions.
736

738 5.2.2 *Learning by Example.* Beyond just making the task faster, the tool also acted as a teaching aid. By pointing out
739 the repetitive code and showing how to fix it, the tool created a "learning moment" exactly when the user needed it.
740 This suggests that automation tools can have two benefits: they help experts work faster, but they also teach beginners
741 difficult concepts—like how to organize blocks of code and use inputs—simply by showing them a practical example.
742

743 5.3 Threats to Validity

744 5.3.1 *Convenience Sampling.* The participants to the study were either acquaintances of one of the authors of the study,
745 or were recruited through these acquaintances. As such, the results of this study do not represent the general population
746 within the domain of chemistry.
747

748 5.3.2 *Limitations to observation.* Due to constraints with time and flexibility, only one of the authors was present to
749 observe the participants. To ensure that data from the observation was not affected by this, a screen recording of each
750 participant performing the task was saved. Several of the authors reviewed and discussed these recordings together to
751 extract data.
752

753 6 Appendices

754 If your work needs an appendix, add it before the "\end{document}" command at the conclusion of your source
755 document.
756

757 Start the appendix with the "appendix" command:
758

759 \appendix
760

761 and note that in the appendix, sections are lettered, not numbered. This document has two appendices, demonstrating
762 the section and subsection identification method.
763

764 7 References

- 765 [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
- 766 [2] Len Bass, Paul Clements, and Rick Kazman. 2021. *Software Architecture in Practice, 4th Edition*. Addison-Wesley Professional.
- 767 [3] Alexander Bock and Ulrich Frank. 2021. Low-Code Platform. *Business and Information Systems Engineering* 63 (2021). doi:10.1007/s12599-021-00726-8
- 768 [4] Ilirian Ibrahim and Dimitris Moudilos. 2022. Towards model reuse in low-code development platforms based on knowledge graphs. In *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings* (Montreal, Quebec, Canada) (MODELS '22). Association for Computing Machinery, New York, NY, USA, 826–836. doi:10.1145/3550356.3561570
- 769 [5] Yuhan Lin and David Weintrop. 2021. The Landscape of Block-Based Programming: Characteristics of Block-Based Environments and How They
770 Support the Transition to Text-Based Programming. *Journal of Computer Languages* 67 (2021), 101075. doi:10.1016/j.cola.2021.101075
- 771 [6] Hugo Lourenço, Carla Ferreira, and João Costa Seco. 2021. OSTRICH - A Type-Safe Template Language for Low-Code Development. In *2021 ACM/IEEE
772 24th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. 216–226. doi:10.1109/MODELS50736.2021.00030
- 773 [7] Vlad Magdalin. 2012. Low code platform tool Webflow. <https://webflow.com/>.

781 [8] Derek Roos. 2005. Low code platform tool Mendix. <https://www.mendix.com/>.
782 [9] Paulo Rosado. 2011. Low code platform tool Outsystems. <https://www.outsystems.com/>.

783
784 Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009
785

786

787

788

789

790

791

792

793

794

795

796

797

798

799

800

801

802

803

804

805

806

807

808

809

810

811

812

813

814

815

816

817

818

819

820

821

822

823

824

825

826

827

828

829

830

831

832 Manuscript submitted to ACM