

1 textOsFtextTOSfliningLFliningTLFtextosflininglftabulartabproportionalprop
2 superiorSup
3 su-
4 pe-
5 ri-
6 or-
7 Sup
8
9
10 fontspechyperref
11
12 **Title 3**
13
14 ANNE-MARIE ROMMERDAHL, SDU, Denmark
15 JEREMY ALEXANDER RAMÍREZ GALEOTTI, SDU, Denmark
16 DIMITRIOS DAFNIS, SDU, Denmark
17 NASIFA AKTER, SDU, Denmark
18 MOHAMMAD HOSEIN KARDOUNI, SDU, Denmark
19 BEN TROVATO* and G.K.M. TOBIN*, Institute for Clarity in Documentation, USA
20 LARS THØRVÄLD, The Thørväld Group, Iceland
21 VALERIE BÉRANGER, Inria Paris-Rocquencourt, France
22
23 A clear and well-documented L^AT_EX document is presented as an article formatted for publication by ACM in a conference proceedings or journal publication. Based on the "acmart" document class, this article presents and explains many of the common variations, as well as many of the formatting elements an author may use in the preparation of the documentation of their work.
24
25 CCS Concepts: • **Do Not Use This Code → Generate the Correct Terms for Your Paper**; *Generate the Correct Terms for Your Paper*; Generate the Correct Terms for Your Paper; Generate the Correct Terms for Your Paper.
26 Additional Key Words and Phrases: Do, Not, Use, This, Code, Put, the, Correct, Terms, for, Your, Paper
27
28 **ACM Reference Format:**
29 Anne-Marie Rommerdahl, Jeremy Alexander Ramírez Galeotti, Dimitrios Dafnis, Nasifa Akter, Mohammad Hosein Kardouni, Ben Trovato, G.K.M. Tobin, Lars Thørväld, and Valerie Béranger. 2018. Title 3. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation email (Conference acronym 'XX)*. ACM, New York, NY, USA, 15 pages. <https://doi.org/XXXXXXX>.
30 XXXXXXXX
31
32
33
34
35
36
37
38
39
40
41 **1 Introduction**
42 ACM's consolidated article template, introduced in 2017, provides a consistent L^AT_EX style for use across ACM publications, and incorporates accessibility and metadata-extraction functionality necessary for future Digital Library endeavors. Numerous ACM and SIG-specific L^AT_EX templates have been examined, and their unique features incorporated into this single new template.
43
44
45
46
47
48 **2 Background and Related Work**
49 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" [?] - i.e. the degree to which the application or its components can be reused. There are multiple¹ 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).
50
51
52 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[?]. This lack of focus on promoting reuse may

53 the top nine most used production-ready screen templates from OutSystems. The end-user may not create and save
 54 their own templates, nor can they re-apply a template which they have customized.
 55

56 Another approach focused on enabling reuse of models, by converting and merging models into a single graph (the
 57 Knowledge Graph), which acts as a repository of models[?]. This graph is used to provide recommendations to the
 58 end-user, based on the model they're currently building. While this feature of recommending models (either constructed
 59 by domain experts and then developed by model experts, or made by the end-user themselves) could prove very useful,
 60 the study is clearly not focused on guiding the user towards reusing their own models.
 61

62 Building on the ideas discussed for improving reuse in low-code development platforms (LCDPs), several popular
 63 tools show these concepts in action. For instance, Webflow[?] is a leading low-code platform that offers a wealth of
 64 features for building responsive websites. One of its standout features is the ability to create reusable components and
 65 UI kits, which can significantly speed up the development process. With Webflow's intuitive interface, developers can
 66 quickly design and prototype components, and then reuse them across multiple pages and projects. Despite all of the
 67 useful features that this tool has, it does not provide guidance to the end-users to create custom reusable components.
 68

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

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

89 In order to analyze how block-based robotics environments address reuse area, 4 representative platforms were
 90 compared: mBlock, MakeCode, SPIKE LEGO, VEXcode GO and Open Roberta. The comparison focused on three main
 91 dimensions of reuse: structural reuse (through user-defined blocks or functions), social reuse (through sharing or
 92 remixing existing projects), and interoperable reuse (through import/export capabilities).
 93

Table 1. Block Based Robotics Environments Reuse Support

Platform	Structural Reuse	Social Reuse	Interoperable Reuse	Reuse Support
VEXcode GO	X	X		Medium
mBlock	X	X	X	Medium
MakeCode	X	X	X	Medium
Spike Lego	X		X	Low
Open Roberta		X		Low

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

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

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. In contrast, our work emphasizes guided abstraction, helping users understand and practice modular design directly within block-based environments.

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

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

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

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

166 3 Study Design

167 Following the Design Science methodology, our study is structured into three main phases: problem investigation to
168 define goals, treatment design to specify the artifact requirements, and treatment validation to assess the artifact's
169 performance in a controlled environment.
170

171 3.1 Problem Investigation

172 3.1.1 *Problem Context and Motivation.* End-user development (EUD) for collaborative robots (cobots) presents unique
173 challenges, particularly for users without formal programming training. In domains such as chemistry laboratories,
174 educational robotics, and industrial settings, end-users need to program robots to perform specific tasks but often lack
175 the software engineering knowledge to write maintainable, well-structured code. In the domain of Chemistry, one of
176 the most relevant and important tasks is performing experiments in labs in order to test a hypothesis, or to aid in the
177 understanding of how chemicals react. Robots can be used in chemistry labs to automate experiments with great effect,
178 as many experiments involve steps that are repetitive, and susceptible to human error, such as a step being overlooked,
179 instructions being misread, etc. Automation of menial tasks will leave the chemists with more time for other work,
180 and also comes with the added bonus of chemists not having to handle dangerous chemicals. One critical challenge in
181 EUD is code reuse. Users frequently create repetitive code because they struggle to recognize duplicate patterns, lack
182 knowledge about abstraction mechanisms, or find existing tools too complex to use effectively. This problem manifests
183 in several ways: programs become unnecessarily long and difficult to maintain and small changes require modifications
184 in multiple locations, increasing the risk of errors. Several visual programming environments, like OpenRoberta Lab,
185 don't provide assistance in identifying when code should be reused or how to extract repeated sequences into reusable
186 components. As lab work in chemistry involves many repetitive tasks, these challenges can easily become an obstacle
187 for the chemists, which may turn them away from using cobots, as the inconvenience outweighs the benefits.
188

189 3.1.2 *Stakeholder Analysis.* Chemists and lab technicians who use cobots for repetitive tasks such as sample prepa-
190 ration, dispensing, mixing, and quality control procedures. They possess deep domain expertise in chemistry but
191 limited programming knowledge, often creating long, repetitive programs that become difficult to maintain when
192 adapting experimental protocols. Their primary need is to quickly create and modify robot programs without becoming
193 programming experts.
194

195 3.2 Treatment Design

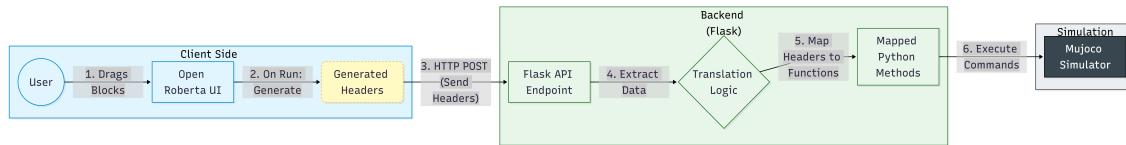
196 To address the problem of code reuse in EUD for cobots, we have derived a set of requirements designed to contribute
197 to the chemist's goal of creating maintainable and reusable robot programs. Functionally, the artifact must be capable
198 of automatically detecting duplicate or similar block sequences and visually highlighting these duplications within
199 Manuscript submitted to ACM
200
201

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

219 3.2.1 *Artifact Specification: The Reuse Assistant.* To satisfy the requirements above, we designed the Reuse Assistant as
 220 an extension of Open Roberta Lab.
 221

222 3.2.2 *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

- 225 (1) **Client Side (Open Roberta):** The user interacts with the Open Roberta UI to assemble block sequences. The
 226 Reuse Assistant operates at this layer, analyzing blocks in real-time. Upon execution, the client generates specific
 227 data structures ("Generated Headers") representing the program logic.
 228
- 229 (2) **Backend (Flask Server):** The client transmits these headers via HTTP POST requests to a Flask-based API
 230 Endpoint. A "Translator" component processes the data, mapping the abstract block definitions to concrete
 231 Python methods compatible with the robot's control logic.
 232
- 233 (3) **Simulation (Mujoco):** The mapped methods trigger the execution of commands within the Mujoco Simulator,
 234 which renders the physical behavior of the cobot in the virtual environment.
 235



236 Fig. 1. System architecture showing the execution flow from the Client (Open Roberta UI) through the Flask Backend to the Mujoco
 237 Simulator.
 238

239 3.2.3 *Detection Algorithm.* The core of the reuse assistance, located on the Client Side, is the sequence detection
 240 algorithm encapsulated in the `highlightOnlyFunctionCandidates` function. The algorithm operates in several steps:
 241

- 242 • **Linearization:** It first converts the hierarchical block structure into a linear chain of significant operational
 243 blocks, filtering out simple literals to focus on logic and action blocks.
 244
- 245 • **Signature Generation:** For a sliding window of block sequences (ranging from a minimum to a maximum
 246 length), it generates a unique "structural signature." This signature is a hash or string representation of the
 247 block types and their connectivity, ignoring specific parameter values.
 248
- 249 • **Pattern Matching:** The algorithm aggregates sequences with identical signatures. If a signature appears more
 250 than once (frequency ≥ 2), it is flagged as a candidate for reuse.
 251

- 261 • **Parameter Extraction:** Once a duplicate group is identified, the `extractLiteralParameters` function compares the instances to identify varying literals. These variations are mapped to future function parameters, ensuring the created abstraction is generalized correctly.
- 262
263
264

Algorithm 1 Sequence Detection and Signature Generation

Require: `Blocks`: List of blocks in workspace

Ensure: `Candidates`: List of duplicate sequences

```

270   1: LinearChain  $\leftarrow$  Linearize(Blocks) {Filter out unconnected literals}
271   2: Signatures  $\leftarrow$  Map<String, List<Sequence>>
272   3: for len  $\leftarrow$  MinLen to MaxLen do
273     4:   for i  $\leftarrow$  0 to length(LinearChain) - len do
274       5:         seq  $\leftarrow$  LinearChain[i : i + len]
275       6:         sig  $\leftarrow$  ""
276       7:         for all block in seq do
277           8:             sig  $\leftarrow$  sig + block.type + "|" {Ignore parameter values}
278         9:         end for
280        10:        Signatures[sig].add(seq)
282      11:      end for
283    12:    end for
285  13:  Candidates  $\leftarrow$  []
286  14:  for all sig in Signatures do
287    15:    if size(Signatures[sig])  $\geq$  2 then
288      16:      Candidates.add(Signatures[sig])
289    17:    end if
291  18:  end for
293  19:  return Candidates

```

295 Algorithm 1. illustrates the core logic for identifying reusable candidates by abstracting literal values.

296

297 3.2.4 *User Interface and Interaction.* The user interface is designed to be intuitive and non-disruptive. When the
298 detection algorithm identifies a candidate, the system visually highlights the blocks on the canvas. A non-blocking toast
299 notification appears, prompting the user to confirm the refactoring. If confirmed, the system automatically generates
300 the custom block definition in a dedicated workspace area (handling visibility via `revealDefinitionWorkspacePane`)
301 and updates the main workspace, replacing the redundant code with concise function calls. This process abstracts the
302 complexity of manual function creation, guiding the user toward modular design practices.
303

304

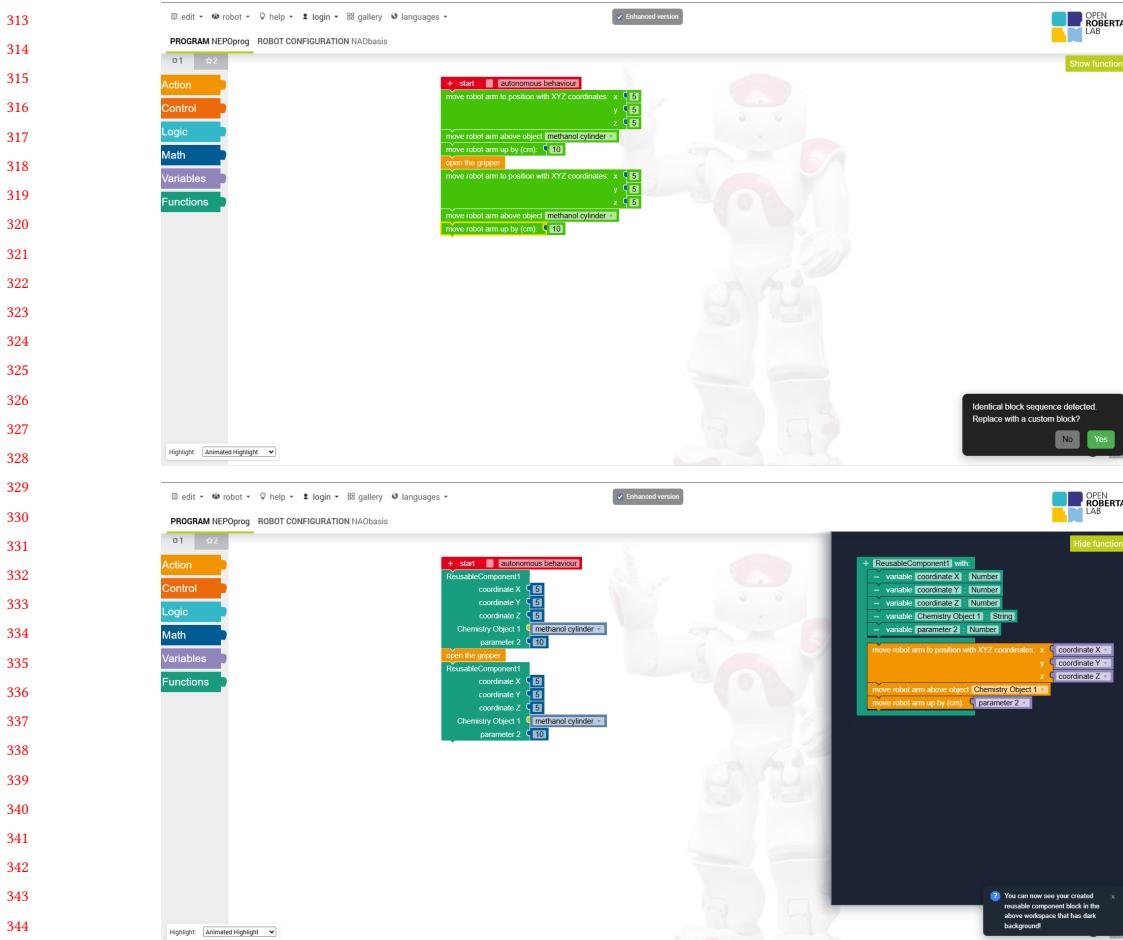


Fig. 2. Guided Reuse Assistant workflow: (Top) The interface detects and highlights duplicate blocks in green. (Bottom) The automated refactoring result, showing the new custom block definition and the simplified main program.

3.3 Treatment Validation

3.3.1 Data Gathering and Analysis. 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 reusable custom blocks within the OpenRoberta environment. Participants will be recruited from local educational institutions, specifically chemistry students and teachers who frequently engage in laboratory work. A sufficient number of (x) participants will be selected to ensure a diverse range of experience levels with block-based programming.

A pre-experiment survey/interview** will be used to gather data about the participants' demographic, and their understanding of modular programming concepts. This is followed by two tasks to be done in the OpenRoberta Lab, designed to make the user focus on reuse. The experimental setup will take place in a controlled environment, where participants will be divided into two groups: one using the enhanced OpenRoberta platform with guided block creation features, and the other using the standard version without these enhancements. Participants' interactions with the

365 platform will be observed throughout the experiment. Data collection will include both quantitative measures, such as
366 task completion time and accuracy in creating reusable blocks and qualitative feedback obtained through a post-task
367 interview. For the qualitative feedback, both groups will have to repeat the task, with the group that initially used the
368 enhanced OpenRoberta platform now using the standard version, while the other group will use the enhanced version.
369 The analysis will compare performance metrics between the two groups and apply thematic analysis to the qualitative
370 data to identify user experiences and perceptions of the new features' usability and effectiveness. This comprehensive
371 evaluation will provide a detailed understanding of how useful and effective is the block creation guidance feature to
372 the end-users.
373

374 3.3.2 *Participant Recruitment.* The participants will be chemistry students and one supervisor from the University
375 of Southern Denmark (SDU). One of the authors of this paper knows a student from the chemistry line whom was
376 recruited for the experiment. This student also assisted in recruiting others from his class. It should be noted, that this
377 selection of participants classifies as a convenience sampling. As such, they may not represent the general population.
378

379 The participants will be asked to fill out a survey before starting the tasks, in order to asses their background, as well
380 as their knowledge about block-based programming, the use of cobots, OpenRoberta Lab and their experience with
381 programming. The survey can be found in appendix XXX.
382

383 3.3.3 *Task Execution.* Before the tasks, the participants will be given a short introduction to the OpenRoberta Lab, as
384 well as the cobot simulator. The participants will then perform two tasks, each task described by a set of pre-defined
385 steps to perform. The first task will be generic in nature. The purpose of this task is to make the user more familiar
386 with block-based programming and the OpenRoberta Lab.
387

388 The second task is more focused on the domain of chemistry, as it is modelled after a real lab experiment perfomed
389 by chemistry students at SDU (appendix XXX). The experiment instructions were obtained from one of the participants.
390 The instructions for both tasks can be found in appendix XXX.
391

392 **4 Modifications**

393 Modifying the template – including but not limited to: adjusting margins, typeface sizes, line spacing, paragraph and
394 list definitions, and the use of the \vspace command to manually adjust the vertical spacing between elements of your
395 work – is not allowed.
396

397 **Your document will be returned to you for revision if modifications are discovered.**
398

399 **5 Typefaces**

400 The “acmart” document class requires the use of the “Libertine” typeface family. Your TeX installation should include
401 this set of packages. Please do not substitute other typefaces. The “lmodern” and “ltimes” packages should not be used,
402 as they will override the built-in typeface families.
403

404 **6 Title Information**

405 The title of your work should use capital letters appropriately - <https://capitalizemytitle.com/> has useful rules for
406 capitalization. Use the `title` command to define the title of your work. If your work has a subtitle, define it with the
407 `subtitle` command. Do not insert line breaks in your title.
408

409 If your title is lengthy, you must define a short version to be used in the page headers, to prevent overlapping text.
410 The `title` command has a “short title” parameter:
411

412 Manuscript submitted to ACM
413

417 \title[short title]{full title}
418
419

420 7 Authors and Affiliations

421 Each author must be defined separately for accurate metadata identification. As an exception, multiple authors may
422 share one affiliation. Authors' names should not be abbreviated; use full first names wherever possible. Include authors'
423 e-mail addresses whenever possible.

424 Grouping authors' names or e-mail addresses, or providing an "e-mail alias," as shown below, is not acceptable:

425
426
427 \author{Brooke Aster, David Mehldau}
428 \email{dave,judy,steve@university.edu}
429 \email{firstname.lastname@phillips.org}

430
431 The authornote and authornotemark commands allow a note to apply to multiple authors — for example, if the
432 first two authors of an article contributed equally to the work.

433 If your author list is lengthy, you must define a shortened version of the list of authors to be used in the page headers,
434 to prevent overlapping text. The following command should be placed just after the last \author{} definition:

435
436
437 \renewcommand{\shortauthors}{McCartney, et al.}

438 Omitting this command will force the use of a concatenated list of all of the authors' names, which may result in
439 overlapping text in the page headers.

440 The article template's documentation, available at <https://www.acm.org/publications/proceedings-template>, has a
441 complete explanation of these commands and tips for their effective use.

442 Note that authors' addresses are mandatory for journal articles.

443 8 Rights Information

444 Authors of any work published by ACM will need to complete a rights form. Depending on the kind of work, and the
445 rights management choice made by the author, this may be copyright transfer, permission, license, or an OA (open
446 access) agreement.

447 Regardless of the rights management choice, the author will receive a copy of the completed rights form once it
448 has been submitted. This form contains L^AT_EX commands that must be copied into the source document. When the
449 document source is compiled, these commands and their parameters add formatted text to several areas of the final
450 document:

- 451
452
453 • the "ACM Reference Format" text on the first page.
454 • the "rights management" text on the first page.
455 • the conference information in the page header(s).

456 Rights information is unique to the work; if you are preparing several works for an event, make sure to use the
457 correct set of commands with each of the works.

458 The ACM Reference Format text is required for all articles over one page in length, and is optional for one-page
459 articles (abstracts).

469 9 CCS Concepts and User-Defined Keywords

470 Two elements of the “acmart” document class provide powerful taxonomic tools for you to help readers find your work
 471 in an online search.

472 The ACM Computing Classification System — <https://www.acm.org/publications/class-2012> — is a set of classifiers
 473 and concepts that describe the computing discipline. Authors can select entries from this classification system, via
 474 <https://dl.acm.org/ccs/ccs.cfm>, and generate the commands to be included in the *L^AT_EX* source.

475 User-defined keywords are a comma-separated list of words and phrases of the authors’ choosing, providing a more
 476 flexible way of describing the research being presented.

477 CCS concepts and user-defined keywords are required for all articles over two pages in length, and are optional
 478 for one- and two-page articles (or abstracts).

483 10 Sectioning Commands

484 Your work should use standard *L^AT_EX* sectioning commands: \section, \subsection, \subsubsection, \paragraph,
 485 and \ subparagraph. The sectioning levels up to \subsubsection should be numbered; do not remove the numbering
 486 from the commands.

487 Simulating a sectioning command by setting the first word or words of a paragraph in boldface or italicized text is
 488 **not allowed**.

489 Below are examples of sectioning commands.

493 10.1 Subsection

495 This is a subsection.

497 10.1.1 Subsubsection. This is a subsubsection.

499 *Paragraph*. This is a paragraph.

501 Subparagraph This is a subparagraph.

503 11 Tables

505 The “acmart” document class includes the “booktabs” package — <https://ctan.org/pkg/booktabs> — for preparing
 506 high-quality tables.

507 Table captions are placed *above* the table.

509 Because tables cannot be split across pages, the best placement for them is typically the top of the page nearest
 510 their initial cite. To ensure this proper “floating” placement of tables, use the environment **table** to enclose the table’s
 511 contents and the table caption. The contents of the table itself must go in the **tabular** environment, to be aligned
 512 properly in rows and columns, with the desired horizontal and vertical rules. Again, detailed instructions on **tabular**
 513 material are found in the *L^AT_EX User’s Guide*.

515 Immediately following this sentence is the point at which Table 2 is included in the input file; compare the placement
 516 of the table here with the table in the printed output of this document.

518 To set a wider table, which takes up the whole width of the page’s live area, use the environment **table*** to enclose
 519 the table’s contents and the table caption. As with a single-column table, this wide table will “float” to a location deemed

Table 2. Frequency of Special Characters

Non-English or Math	Frequency	Comments
\emptyset	1 in 1,000	For Swedish names
π	1 in 5	Common in math
\$	4 in 5	Used in business
Ψ_1^2	1 in 40,000	Unexplained usage

Table 3. Some Typical Commands

Command	A Number	Comments
\author	100	Author
\table	300	For tables
\table*	400	For wider tables

more desirable. Immediately following this sentence is the point at which Table 3 is included in the input file; again, it is instructive to compare the placement of the table here with the table in the printed output of this document.

Always use midrule to separate table header rows from data rows, and use it only for this purpose. This enables assistive technologies to recognise table headers and support their users in navigating tables more easily.

12 Math Equations

You may want to display math equations in three distinct styles: inline, numbered or non-numbered display. Each of the three are discussed in the next sections.

12.1 Inline (In-text) Equations

A formula that appears in the running text is called an inline or in-text formula. It is produced by the **math** environment, which can be invoked with the usual `\begin{...} \end{...}` construction or with the short form `$...$`. You can use any of the symbols and structures, from α to ω , available in L^AT_EX [?]; this section will simply show a few examples of in-text equations in context. Notice how this equation: $\lim_{n \rightarrow \infty} x = 0$, set here in in-line math style, looks slightly different when set in display style. (See next section).

12.2 Display Equations

A numbered display equation—one set off by vertical space from the text and centered horizontally—is produced by the **equation** environment. An unnumbered display equation is produced by the **displaymath** environment.

Again, in either environment, you can use any of the symbols and structures available in L^AT_EX; this section will just give a couple of examples of display equations in context. First, consider the equation, shown as an inline equation above:

$$\lim_{n \rightarrow \infty} x = 0 \tag{1}$$

Notice how it is formatted somewhat differently in the **displaymath** environment. Now, we'll enter an unnumbered equation:

$$\sum_{i=0}^{\infty} x + 1$$

⁵⁷³ and follow it with another numbered equation:

$$\sum_{i=0}^{\infty} x_i = \int_0^{\pi+2} f \quad (2)$$

⁵⁷⁷ just to demonstrate L^AT_EX's able handling of numbering.

⁵⁸⁰ 13 Figures

⁵⁸¹ The “figure” environment should be used for figures. One or more images can be placed within a figure. If your figure
⁵⁸² contains third-party material, you must clearly identify it as such, as shown in the example below.



⁶¹⁷ Fig. 3. 1907 Franklin Model D roadster. Photograph by Harris & Ewing, Inc. [Public domain], via Wikimedia Commons. (<https://goo.gl/VLCRBB>)

⁶²¹ Your figures should contain a caption which describes the figure to the reader.

⁶²² Figure captions are placed *below* the figure.

625 Every figure should also have a figure description unless it is purely decorative. These descriptions convey what's in
 626 the image to someone who cannot see it. They are also used by search engine crawlers for indexing images, and when
 627 images cannot be loaded.

628 A figure description must be unformatted plain text less than 2000 characters long (including spaces). **Figure**
 629 **descriptions should not repeat the figure caption – their purpose is to capture important information that is**
 630 **not already provided in the caption or the main text of the paper.** For figures that convey important and complex
 631 new information, a short text description may not be adequate. More complex alternative descriptions can be placed in
 632 an appendix and referenced in a short figure description. For example, provide a data table capturing the information in
 633 a bar chart, or a structured list representing a graph. For additional information regarding how best to write figure
 634 descriptions and why doing this is so important, please see <https://www.acm.org/publications/taps/describing-figures/>.

635 13.1 The “Teaser Figure”

636 A “teaser figure” is an image, or set of images in one figure, that are placed after all author and affiliation information,
 637 and before the body of the article, spanning the page. If you wish to have such a figure in your article, place the
 638 command immediately before the `\maketitle` command:

```
639 \begin{teaserfigure}
640   \includegraphics[width=\textwidth]{sampleteaser}
641   \caption{figure caption}
642   \Description{figure description}
643 \end{teaserfigure}
```

644 14 Citations and Bibliographies

645 The use of Bib_TE_X for the preparation and formatting of one's references is strongly recommended. Authors' names
 646 should be complete — use full first names (“Donald E. Knuth”) not initials (“D. E. Knuth”) — and the salient identifying
 647 features of a reference should be included: title, year, volume, number, pages, article DOI, etc.

648 The bibliography is included in your source document with these two commands, placed just before the `\end{document}`
 649 command:

```
650 \bibliographystyle{ACM-Reference-Format}
651 \bibliography{bibfile}
```

652 where “`bibfile`” is the name, without the “`.bib`” suffix, of the Bib_TE_X file.

653 Citations and references are numbered by default. A small number of ACM publications have citations and references
 654 formatted in the “author year” style; for these exceptions, please include this command in the **preamble** (before the
 655 command “`\begin{document}`”) of your L_AT_EX source:

```
656 \citetstyle{acmauthoryear}
```

657 Some examples. A paginated journal article [?], an enumerated journal article [?], a reference to an entire issue [?],
 658 a monograph (whole book) [?], a monograph/whole book in a series (see 2a in spec. document) [?], a divisible-book
 659 such as an anthology or compilation [?] followed by the same example, however we only output the series if the volume
 660 number is given [?] (so Editor00a's series should NOT be present since it has no vol. no.), a chapter in a divisible book
 661 [?], a chapter in a divisible book in a series [?], a multi-volume work as book [?], a couple of articles in a proceedings
 662 (of a conference, symposium, workshop for example) (paginated proceedings article) [? ?], a proceedings article with
 663

677 all possible elements [?], an example of an enumerated proceedings article [?], an informally published work [?],
 678 a couple of preprints [? ?], a doctoral dissertation [?], a master's thesis: [?], an online document / world wide web
 679 resource [? ? ?], a video game (Case 1) [?] and (Case 2) [?] and [?] and (Case 3) a patent [?], work accepted for
 680 publication [?], 'YYYYb'-test for prolific author [?] and [?]. Other cites might contain 'duplicate' DOI and URLs (some
 681 SIAM articles) [?]. Boris / Barbara Beeton: multi-volume works as books [?] and [?]. A presentation [?]. An article
 682 under review [?]. A couple of citations with DOIs: [? ?]. Online citations: [? ? ?]. Artifacts: [?] and [?].
 683
 684

685 15 Acknowledgments

686 Identification of funding sources and other support, and thanks to individuals and groups that assisted in the research
 687 and the preparation of the work should be included in an acknowledgment section, which is placed just before the
 688 reference section in your document.
 689

690 This section has a special environment:

```
691 \begin{acks}  

692 ...  

693 \end{acks}
```

694 so that the information contained therein can be more easily collected during the article metadata extraction phase, and
 695 to ensure consistency in the spelling of the section heading.

696 Authors should not prepare this section as a numbered or unnumbered \section; please use the "acks" environment.
 697

701 16 Appendices

702 If your work needs an appendix, add it before the "\end{document}" command at the conclusion of your source
 703 document.
 704

705 Start the appendix with the "appendix" command:

```
706 \appendix
```

707 and note that in the appendix, sections are lettered, not numbered. This document has two appendices, demonstrating
 708 the section and subsection identification method.
 709

712 17 Multi-language papers

713 Papers may be written in languages other than English or include titles, subtitles, keywords and abstracts in different
 714 languages (as a rule, a paper in a language other than English should include an English title and an English abstract).
 715 Use language=... for every language used in the paper. The last language indicated is the main language of the paper.
 716 Use language=english for the English title and abstract.
 717 For example, a French paper with additional titles and abstracts in English and German may start with the following
 718 command
 719

```
720 \documentclass[sigconf, language=english, language=german,  

721 language=french]{acmart}
```

722 The title, subtitle, keywords and abstract will be typeset in the main language of the paper. The commands
 723 \translatedXXX, XXX begin title, subtitle and keywords, can be used to set these elements in the other languages. The
 724 environment translatedabstract is used to set the translation of the abstract. These commands and environment have
 725
 726 Manuscript submitted to ACM

729 a mandatory first argument: the language of the second argument. See `sample-sigconf-i13n.tex` file for examples of
730 their usage.
731

732 18 SIGCHI Extended Abstracts

733 The “sigchi-a” template style (available only in L^AT_EX and not in Word) produces a landscape-orientation formatted
734 article, with a wide left margin. Three environments are available for use with the “sigchi-a” template style, and
735 produce formatted output in the margin:
736

737 **sidebar:** Place formatted text in the margin.

738 **marginfigure:** Place a figure in the margin.

739 **maintable:** Place a table in the margin.

740 Acknowledgments

741 To Robert, for the bagels and explaining CMYK and color spaces.

742 A Research Methods

743 A.1 Part One

744 Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi malesuada, quam in pulvinar varius, metus nunc
745 fermentum urna, id sollicitudin purus odio sit amet enim. Aliquam ullamcorper eu ipsum vel mollis. Curabitur quis
746 dictum nisl. Phasellus vel semper risus, et lacinia dolor. Integer ultricies commodo sem nec semper.
747

748 A.2 Part Two

749 Etiam commodo feugiat nisl pulvinar pellentesque. Etiam auctor sodales ligula, non varius nibh pulvinar semper.
750 Suspendisse nec lectus non ipsum convallis congue hendrerit vitae sapien. Donec at laoreet eros. Vivamus non purus
751 placerat, scelerisque diam eu, cursus ante. Etiam aliquam tortor auctor efficitur mattis.
752

753 B Online Resources

754 Nam id fermentum dui. Suspendisse sagittis tortor a nulla mollis, in pulvinar ex pretium. Sed interdum orci quis metus
755 euismod, et sagittis enim maximus. Vestibulum gravida massa ut felis suscipit congue. Quisque mattis elit a risus ultrices
756 commodo venenatis eget dui. Etiam sagittis eleifend elementum.
757

758 Nam interdum magna at lectus dignissim, ac dignissim lorem rhoncus. Maecenas eu arcu ac neque placerat aliquam.
759 Nunc pulvinar massa et mattis lacinia.
760

761 Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009
762