Authoring Worked Examples for Java Programming with Human-AI Collaboration

Mohammad Hassany
moh70@pitt.edu
University of Pittsburgh
Pittsburgh, PA, USA

Jiaze Ke
jiazek@andrew.cmu.edu
Carnegie Mellon University
Pittsburgh, PA, USA

Peter Brusilovsky
peterb@pitt.edu
University of Pittsburgh
Pittsburgh, PA, USA

Arun Balajiee Lekshmi Narayanan
arl122@pitt.edu
University of Pittsburgh
Pittsburgh, PA, USA

Kamil Akhuseyinoglu
kaa108@pitt.edu
University of Pittsburgh
Pittsburgh, PA, USA

ABSTRACT

Worked examples are among the most popular types of learning content in programming classes. However, instructors rarely have time to provide line-by-line explanations for a large number of examples typically used in a programming class. In this paper, we explore and assess a human-AI collaboration approach to authoring worked examples for Java programming. We introduce an authoring system for creating Java worked examples that generates a starting version of code explanations and presents it to the instructor to edit if necessary. We also present a study that assesses the quality of explanations created with this approach.

CCS CONCEPTS

• **Computing methodologies** → **Natural language processing**; • **Social and professional topics** → **CS1**.

KEYWORDS

Code Examples, Authoring Tool, Human-AI Collaboration

ACM Reference Format:
Mohammad Hassany, Jiaze Ke, Peter Brusilovsky, Arun Balajiee Lekshmi Narayanan, and Kamil Akhuseyinoglu. 2024. Authoring Worked Examples for Java Programming with Human-AI Collaboration. In *The 39th ACM/SIGAPP Symposium on Applied Computing (SAC '24), April 8–12, 2024, Avila, Spain.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3605098.3636160

1 INTRODUCTION

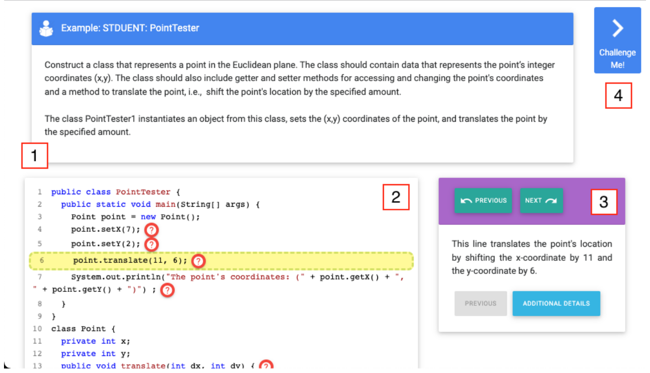
Program code examples known also as *worked examples* play a crucial role in learning how to program [7]. A typical worked example presents a code for solving a specific programming problem and explains the role and function of code lines or code chunks. In textbooks, these explanations are usually presented as comments in the code or as explanations on the margins. which is known for its low efficiency. Recognizing that this approach is not engaging, several research teams developed dedicated learning tools that offered more interactive and engaging ways to learn from examples [1, 4, 6].

A modern interactive tool for studying code examples, such as PCEX system [4] provides interactive access to code examples augmented with instructor explanations, enabling students to selectively study explanations for code fragments they want. Separating explanations from the code keeps the code intact and allows to study explanations at several levels of detail (Figure 1).

PCEX and similar example-focused tools demonstrated their effectiveness in classroom studies, but their practical impact, i.e., broader use by instructors was limited due to the *authoring bottleneck*. Although the creators of example-focused learning tools such as PCEX usually provide a good set of worked examples that can be presented through their tools, many instructors prefer to use their own favorite code examples. The instructors are usually happy to broadly share the code of examples they created (usually adding it to the course Web page), but they rarely have time or patience to augment their favorite examples with detailed line-by-line explanations.

Several approaches have been explored in the past to resolve this authoring bottleneck. For example, learner-sourcing approach engaged students in creating and reviewing explanations for instructor-provided code [5] while NLP was used for automatic extraction of explanations from available sources, such as lecture recordings [6]. In this paper, we present an alternative approach to address the authoring bottleneck based on human-AI collaboration. With this approach, instructors provide the code of their favorite examples along with the statements of the programming problem they are solving. The AI engine based on large language models (LLM) examines the code and generates explanations for each code line. The explanations could be reviewed and edited by the instructor. To support and explore this authoring approach, we created an authoring system, which we expect to radically decrease the time to create a new interactive worked example. The system is briefly introduced in the next section.

The main content of the paper is focused on answering the feasibility question, which we consider as the first step in "proving" our approach: to what extent code explanations generated by ChatGPT could be considered satisfactory to serve as the basis for human-AI collaborative authoring. To answer this question, we performed a



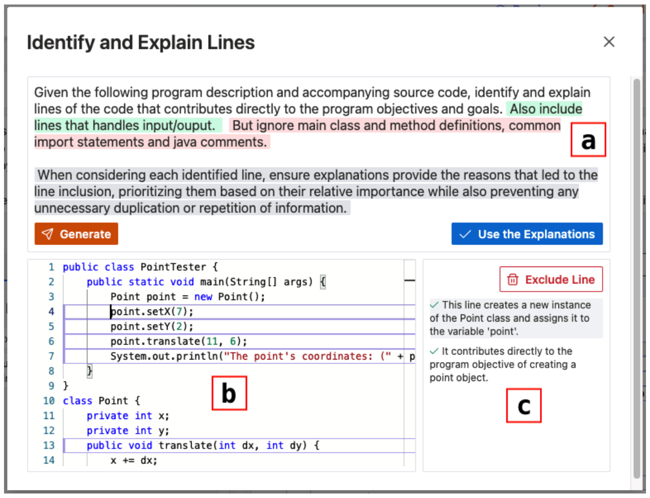
Construct a class that represents a point in the Euclidean plane. The class should contain data that represents the point's integer coordinates (x,y). The class should also include getter and setter methods for accessing and changing the point's coordinates and a method to translate the point, i.e., shift the point's location by the specified amount.

The class PointTester1 instantiates an object from this class, sets the (x,y) coordinates of the point, and translates the point by the specified amount.

```
public class PointTester {
  public static void main(String[] args) {
    Point point = new Point();
    point.setX(7);
    point.setY(2);
    point.translate(11, 6);
    System.out.println("The point's coordinates: (" + point.getX() + ",
" + point.getY() + ")") ;
  }
}
class Point {
  private int x;
  private int y;
  public void translate(int dx, int dy) {
```

This line translates the point's location by shifting the x-coordinate by 11 and the y-coordinate by 6.

Figure 1: Studying a code example in the PCEX system: 1) problem description, 2) source code of problem solution, 3) explanations for the highlighted line.



Given the following program description and accompanying source code, identify and explain lines of the code that contributes directly to the program objectives and goals. Also include lines that handles input/ouput. But ignore main class and method definitions, common import statements and java comments.

When considering each identified line, ensure explanations provide the reasons that led to the line inclusion, prioritizing them based on their relative importance while also preventing any unnecessary duplication or repetition of information.

```
public class PointTester {
    public static void main(String[] args) {
        Point point = new Point();
        point.setX(7);
        point.setY(2);
        point.translate(11, 6);
        System.out.println("The point's coordinates: (" + p
    }
}
class Point {
    private int x;
    private int y;
    public void translate(int dx, int dy) {
        x += dx;
```

✓ This line creates a new instance of the Point class and assigns it to the variable 'point'.

✓ It contributes directly to the program objective of creating a point object.

Figure 2: The explanation generation interface, a) default prompt (author can tune the prompt - optional), b) example code (lines with explanations are marked with a purple border), c) generated explanations for the selected line.

user study in which TAs and students compared code explanations created by experts through a traditional process with examples created by ChatGPT.

2 HUMAN-AI COLLABORATIVE AUTHORING

In the collaborative authoring process supported by our tool, the main task of a human author is to provide the code of the example and the statement of the problem that the code solves. The main task of ChatGPT is to generate the bulk of code line explanations on several levels of detail. As an option, a human author could edit and refine the text produced by ChatGPT to adapt it to the objectives of the class and the target students. As in any productive collaboration, each side does what it is best suited to do, leaving the challenging work to the partner. In the tool interface (Figure 2) the author should click "Generate" to generate the explanations, and then click "Use the Explanations" to add them to the example. The tool provides the human author several opportunities to control the outcome of the explanation generation process: 1) the author can tune the prompt to their needs, 2) the author can decide whether to include or exclude a generated explanation, and 3) the author can edit or remove the explanation after it is edited. More details about the authoring system and its interface could be found in [3]

3 EVALUATION

To assess the quality of the explanations generated by ChatGPT using the best-performing prompt with options tuned through the internal evaluation, we performed a user study. In this study, we compared the explanations generated by ChatGPT with the explanations created by experts for the same PCEX examples. Unlike some earlier studies that used beginner students to evaluate ChatGPT explanations, we used more experienced users, advanced undergraduate and graduate students. The reason for this difference is that in our authoring system, direct users of the ChatGPT explanation are not *consumers* of explanations, but prospective *authors*. In the implemented human-AI collaborative authoring approach, authors have the option to edit the generated explanation. Thus, it is up to the prospective authors to decide how good the explanations are since their perception of quality impacts the amount of their work: poor explanations will require a lot of editing, while good explanations could be accepted as-is or with minimal changes.

We recruited 15 evaluators, six graduate students who usually serve as assistants or instructors in programming classes (we refer to them as *authors*) and 9 undergraduate students who completed a Java programming class. The participants had to provide their responses through an evaluation form and received an Amazon gift card as compensation.

The evaluation form included 8 examples from PCEX system [4], each with a problem description and solution code. For each code line, the form displayed an explanation generated for this line by ChatGPT and by an expert. The participants had to rate both explanations for a given line of code and compare them. The order of ChatGPT and expert explanations for a given line of code was randomized, and the evaluators did not know which explanation was generated by ChatGPT or the expert. The expert explanations for the PCEX examples were written by instructors and polished over several years of classroom use, so they offer a good basis for comparison.

To evaluate the explanations, the participants had to rate to what extent each explanation is *complete* and which is *better*. We defined a *better* explanation as "providing more information, going deeper, better connecting to programming concepts". The participants had to use the following metrics:

(1) *Explanation 1 is sufficiently complete*: Not complete (0), Complete (1), Very complete (2)
(2) *Explanation 2 is sufficiently complete*: Not complete (0), Complete (1), Very complete (2)
(3) *Which explanation is better?* Both are the same (0), Explanation 1 is better (1), Explanation 2 is better (2)

From the collected responses, we excluded lines that had ChatGPT or expert explanations but not both. For the remaining 45 lines of code, we observed from the evaluators' ratings for the question

	Not complete=0	Complete=1	Very complete=2
ChatGPT			
Students	0.00%	13.33%	86.67%
Authors	1.48%	32.59%	65.93%
Overall	0.59%	21.04%	78.37%
Expert			
Students	2.22%	55.56%	42.22%
Authors	14.07%	57.78%	28.15%
Overall	6.96%	56.44%	36.59%

Table 1: Percentage of Ratings for different items on the scale for "Explanation 1 / 2 is sufficiently complete?"

	Explanation		
Rating	Students	Authors	Overall
Both are the same = 0	32.84%	14.44%	25.48%
Expert is better = 1	16.05%	27.41%	20.59%
ChatGPT is better = 2	51.11%	58.15%	53.93%

Table 2: Percentage of Ratings for the different items on the scale for "Which explanation is better?"

"Explanation 1 is sufficiently complete?" or "Explanation 2 is sufficiently complete?" that the ChatGPT explanations were rated as 0.59% (not complete), 21.04% (complete) and 78.37% (very complete) compared to Expert explanations as 6.96% (not complete), 56.44% (complete), and 36.59% (very complete). In response to the question "Which explanation is better?", evaluators selected ChatGPT as the better explanation in 53.93% of lines, compared to experts (20.59%); and in the rest of the lines (25.48%) both were rated the same. Our calculations of the inter-rater reliability for the ratings of the question "Which explanation is better?" using Fleiss-Kappa gave us 0.182, $p < 0.01$ score of agreement. This can be interpreted as "slight agreement" based on the 2-raters/2-categories table. Given that Fleiss-Kappa is a chance-corrected coefficient, it can be interpreted as a better agreement due to the high number of subjects.

We observe that the students did not rate ChatGPT explanations incomplete at all with their (13. 33%) and (86. 67%) ratings, indicating that ChatGPT explanations are complete and very complete, respectively. The authors also rated the ChatGPT explanations as complete (32.59%) or very complete (65.93%). Hence, a majority of authors and students find the ChatGPT explanations complete, as shown in Table 1. In terms of comparing which explanations is better, 51.11% and 58.15% of students and authors, respectively, find that the explanations of ChatGPT are better for the given lines of code. On average, the authors rated the ChatGPT explanations more complete than the students, and the students preferred the ChatGPT explanations more than the authors, as summarized in Table 3. A direct comparison of two options, based on the question "which explanation is better (ChatGPT vs Expert)?", is presented in Table 2. Given that the assessment was performed using blind rating, this is an encouraging result for the use of generative AI for authoring tools.

	All	Students	Authors
ChatGPT*	1.867 (0.133)	1.644 (0.258)	1.778 (0.163)
Expert*	1.400 (0.388)	1.141 (0.465)	1.296 (0.408)
Which is better?	1.183 (0.510)	1.437 (0.373)	1.284 (0.427)

Table 3: Average (Stdev) Ratings - *Completeness

4 CONCLUSION

In this paper, we introduce a worked example authoring tool that utilizes ChatGPT for the automatic generation of line-by-line code explanations and report the results of a study that assessed the quality and completeness of generated explanations. Although there have been several attempts to use LLMs such as OpenAI Codex or ChatGPT [2, 8, 9] to generate code explanations, our work is the first attempt to integrate the power of LLM into an example authoring tool that enables prospective example authors to produce fully explained worked example through human-AI collaboration. To our knowledge, our work is also the first one that compared the quality and completeness of instructor-authored and ChatGPT-generated code explanations from the prospects of both authors and learners. The presented study is the first step in evaluating the value and feasibility of collaborative example authoring. To reliably assess the quality of explanations produced through human-AI collaboration and their value for students, we plan to run a multisemester-long study engaging instructors to use the tool to produce worked examples for their classes.

REFERENCES

[1] Peter Brusilovsky, Michael V. Yudelson, and I-Han Hsiao. 2009. Problem Solving Examples as First Class Objects in Educational Digital Libraries: Three Obstacles to Overcome. *Journal of Educational Multimedia and Hypermedia* 18 (2009), 267–288.
[2] Eason Chen, Ray Huang, Han-Shin Chen, Yuen-Hsien Tseng, and Liang-Yi Li. 2023. GPTutor: A ChatGPT-Powered Programming Tool for Code Explanation. In *Artificial Intelligence in Education*. Springer Nature Switzerland, Cham, 321–327.
[3] Mohammad Hassany, Peter Brusilovsky, Jiaze Ke, Kamil Akhuseyinoglu, and Arun Balajiee Lekshmi Narayanan. 2023. *Authoring Worked Examples for Java Programming with Human-AI Collaboration.* Report arXiv:2312.02105. arXiv. https://doi.org/10.48550/arXiv.2312.02105
[4] Roya Hosseini, Kamil Akhuseyinoglu, Peter Brusilovsky, Lauri Malmi, Kerttu Pollari-Malmi, Christian Schunn, and Teemu Sirkiä. 2020. Improving Engagement in Program Construction Examples for Learning Python Programming. *International Journal of Artificial Intelligence in Education* 30, 2 (01 Jun 2020), 299–336.
[5] I-Han Hsiao and Peter Brusilovsky. 2011. The Role of Community Feedback in the Student Example Authoring Process: an Evaluation of AnnotEx. *British Journal of Educational Technology* 42, 3 (2011), 482–499.
[6] Kandarp Khandwala and Philip J. Guo. 2018. Codemotion: expanding the design space of learner interactions with computer programming tutorial videos. *Proceedings of the Fifth Annual ACM Conference on Learning at Scale* (2018).
[7] Marcia C. Linn and Michael J. Clancy. 1992. The case for case studies of programming problems. *Commun. ACM* 35 (1992), 121–132.
[8] Stephen MacNeil, Andrew Tran, Arto Hellas, Joanne Kim, Sami Sarsa, Paul Denny, Seth Bernstein, and Juho Leinonen. 2023. Experiences from Using Code Explanations Generated by Large Language Models in a Web Software Development E-Book. In *54th ACM Technical Symposium on Computer Science Education V. 1* (Toronto ON, Canada) *(SIGCSE 2023)*. 931–937.
[9] Sami Sarsa, Paul Denny, Arto Hellas, and Juho Leinonen. 2022. Automatic Generation of Programming Exercises and Code Explanations Using Large Language Models. In *2022 ACM Conference on International Computing Education Research - Volume 1 (ICER '22)*. 27–43.