**1.anonymize authors**

**2.CCSXML**

**3.figure out keywords**

# A Domain Analysis of Data Structure and Algorithm Explanations in the Wild

Jeffrey Young
Oregon State University
School of EECS
Corvallis, Oregon, USA

Eric Walkingshaw
Oregon State University
School of EECS
Corvallis, Oregon, USA

## ABSTRACT

Explanations of data structures and algorithms are complex interactions of several notations, including natural language, mathematics, pseudocode, and diagrams. Currently, such explanations are created ad hoc using a variety of tools, and the resulting artifacts are static, reducing explanatory value. We envision a domain-specific language for developing rich, interactive explanations of data structures and algorithms. In this paper, we analyze this domain to sketch requirements for our language. We build on an existing pedagogic theory of explanation, which we adapt to a qualitative coding system for explanation artifacts collected online. We show that explanations of algorithms and data structures in the wild exhibit patterns predicted by the pedagogic theory and derive insights for our language. This work is part of our effort to develop the paradigm of explanation-oriented programming, which shifts the focus of programming from computing results to producing rich explanations of how those results were computed.

## CCS CONCEPTS

• **Computer systems organization → Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability;

## KEYWORDS

ACM proceedings, LaTeX, text tagging

**ACM Reference format:**
Jeffrey Young and Eric Walkingshaw. 2017. A Domain Analysis of Data Structure and Algorithm Explanations in the Wild. In *Proceedings of SIGCSE, Seattle, Washington USA, March 2017 (Seattle'2017),* 5 pages.
https://doi.org/10.475/123_4

## 1 INTRODUCTION

Data structures and algorithms are at the heart of computer science and must be explained to each new generation of students. How can we do this effectively?

In this paper, we focus on the *artifacts* that constitute or support explanations of data structures and algorithms (hereafter just "algorithms"), which can be shared and reused. For verbal explanations, such as a lecture, the supporting artifact might be the associated slides. For written explanations, the artifact is the explanation as a whole, including the text and any supporting figures. Explanation artifacts associated with algorithms are interesting because they typically present a complex interaction among many different notations, including natural language, mathematics, pseudocode, executable code, various kinds of diagrams, animations, and more.

Currently, explanation artifacts for algorithms are created ad hoc using a variety of tools and techniques, and the resulting explanations tend to be static, reducing their explanatory value. Although there has been a substantial amount of work on algorithm visualization [3–7, 10], and tools exist for creating these kinds of supporting artifacts, there is no good solution for creating integrated, multi-notational explanations as a whole. Similarly, although some algorithm visualization tools provide a means for the student to tweak the parameters or inputs to an algorithm to generate new visualizations, they do not support creating cohesive interactive explanations that correspondingly modify the surrounding explanation or that allow the student to respond to or query the explanation in other ways. To fill this gap, we envision a *domain-specific language* (DSL) that supports the creation of rich, interactive, multi-notational artifacts for explaining algorithms. The development of this DSL is part of a larger effort to explore the new paradigm of *explanation-oriented programming*, briefly described in Section **??**.

The intended users of the envisioned DSL are CS educators who want to create *interactive artifacts* to support the explanation of algorithms. These users are experts on the corresponding algorithms, and also trained and skilled programmers. The produced explanation artifacts might supplement a lecture or be posted to a web page as a self-contained (textual and graphical) explanation. The DSL should support pedagogical methods directly through built-in abstractions and language constructs. It should also support a variety of forms of student interaction. For example, teachers should be able to define equivalence relations enabling users to automatically generate variant explanations [2], to build in specific responses to anticipated questions, and to provide explanations at multiple levels of abstraction.

This paper represents a formative step toward this vision. We conduct a *qualitative analysis* of our domain in order to determine the form and content of the explanation artifacts that educators are already creating. We base our analysis on an established pedagogical theory in order to better understand how well existing artifacts support the pedagogical process of explaining a complex topic.

More specifically, we collect 30 explanation artifacts from the internet, consisting of slide decks and lecture notes that explain two algorithms and one data structure commonly covered in undergraduate computer science courses: Dijkstra's shortest path algorithm [8, pp. 137–142], merge sort [8, 210–214], and AVL trees [9, pp. 458–475]. We analyze these artifacts by applying a classic pedagogical

Permission to make digital or hard copies of part or all 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 third-party components of this work must be honored. For all other uses, contact the owner/author(s).
*Seattle'2017, March 2017, Seattle, Washington USA*
© 2017 Copyright held by the owner/author(s).
ACM ISBN 123-4567-24-567/08/06...$15.00
https://doi.org/10.475/123_4

theory by Bellack et al. [1] that describes the patterns of language used in the process of teaching. Bellack et al. define a typology for coding transcripts of teacher and student verbalizations during teaching. An overview of the typology is given in Section **??**.

This paper makes the following contributions:

C1. We adapt Bellack et al.'s typology for analyzing *explanation activities* in the form of classroom discourse to support analyzing *explanation artifacts* for algorithms in the form of lecture notes and slides (Section **??**).

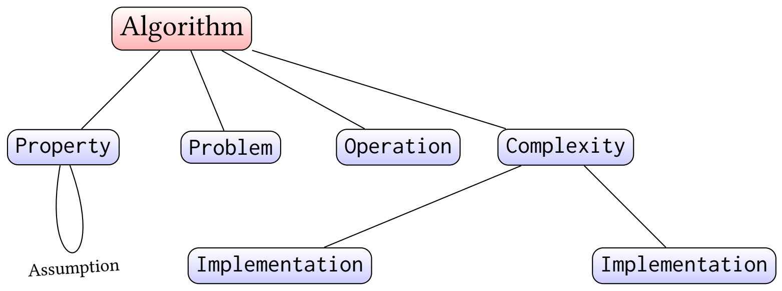
C2. We provide a coded qualitative data set of explanation artifacts, using the system defined in C1 applied to our sample of 30 collected explanation artifacts (Section **??**).

C3. We note that conceptual units called *teaching cycles*, described in the pedagogy literature, can be observed in the collected artifacts. This suggests that the adaptation of Bellack's system to artifacts is valid and that similar teaching strategies are used in explanation artifacts as in verbal explanations (Section **??**).

C4. We further analyze the data set to motivate the need for a DSL and to extract insights that will be useful for the design of such a DSL (Section **??**). For example, we note that the artifacts do not include any steps coded with the *reacting* or *responding* pedagogical moves defined by Bellack et al.'s typology, reflecting the fact that static artifacts treat the learner as an information sink with no recourse for query or response.

C5. We describe how Bellack et al.'s theory can directly provide a semantics basis for a DSL and argue for the advantages of such an approach (Section **??**).

## 2 BACKGROUND



4.Fix this caption

**Figure 1: Explanation Tree for Dijkstra’s 009**

## REFERENCES

[1] Arno A Bellack, Frank L Smith, Herbert M Kliebard, and Ronald T Hyman. 1966. *The Language of the Classroom.* Teachers College Press.

[2] Martin Erwig and Eric Walkingshaw. 2013. A Visual Language for Explaining Probabilistic Reasoning. *Journal of Visual Languages and Computing (JVLC)* 24, 2 (2013), 88–109.

[3] Peter Gloor. 1997. Animated Algorithms. In *Elements of Hypermedia Design: Techniques for Navigation & Visualization in Cyberspace.* Birkhäuser, Boston, 235–241.

[4] Peter A. Gloor. 1992. AACE – Algorithm Animation for Computer Science Education. In *IEEE Workshop on Visual Languages.* 25–31.

[5] Steven Hansen, N.Hari Narayanan, and Mary Hegarty. 2002. Designing Educationally Effective Algorithm Visualizations. *Journal of Visual Languages and Computing (JVLC)* 13, 3 (2002), 291 – 317. https://doi.org/10.1006/jvlc.2002.0236

[6] Christopher D. Hundhausen, Sarah A. Douglas, and John T. Stasko. 2002. A Meta-Study of Algorithm Visualization Effectiveness. *Journal of Visual Languages and Computing (JVLC)* 13 (2002), 259–290.

[7] Charles Kann, Robert W. Lindeman, and Rachelle Heller. 1997. Integrating algorithm animation into a learning environment. *Computers and Education (CE)* 28, 4 (1997), 223 – 228. https://doi.org/10.1016/S0360-1315(97)00015-8

[8] Jon Kleinberg and Eva Tardos. 2006. *Algorithm design.* Pearson Education, Boston.

[9] Donald E. Knuth. 1998. *The Art of Computer Programming: Sorting and Searching* (2nd ed.). Pearson Education, Boston.

[10] Clifford A Shaffer, Matthew L Cooper, Alexander Joel D Alon, Monika Akbar, Michael Stewart, Sean Ponce, and Stephen H Edwards. 2010. Algorithm visualization: The state of the field. *ACM Transactions on Computing Education (TOCE)* 10, 3 (2010), 9.