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

Authors anonymized for review

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.**

I. 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 [**?**], [**?**], [**?**], [**?**], [**?**], [**?**], 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 [**?**], 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 [**?**, pp. 137–142], merge sort [**?**, 210–214], and AVL trees [**?**, pp. 458–475]. We analyze these artifacts by applying a classic pedagogical theory by Bellack et al. [**?**] 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 C**??** 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 **??**).

In Section **??** we discuss related work in the areas of algorithm visualization and argue that visualization alone is an insufficient form of explanation artifact; we also provide an overview of other typologies of explanations and discuss why we chose Bellack et al.'s typology over these alternatives.

II. Background

In this section, we put the present work into context by the describing the paradigm of *explanation-oriented programming* in Section **??**, the exploration of which is an underlying motivation of our work, and by introducing the pedagogical typology of Bellack et al. [**?**] in Section **??**, which is the theoretical foundation of our analysis.

A. Explanation-oriented programming

Explanation-oriented programming (XOP) is a new programming paradigm where the primary output of a program is not a set of computed values, but an *explanation of how* those values were computed [**?**], [**?**], [**?**], [**?**], [**?**]. A high-level goal of this work is to further develop the paradigm of XOP through the development of a specific DSL.

Programming languages for XOP should not merely produce explanations as a byproduct, but should provide abstractions and features specific to the creation of interactive explanation artifacts. For example, they should provide facilities for creating application-specific notations and visualizations (which are widespread in explanations of algorithms), and for describing alternative explanations produced in response to user input, for example, at different levels of abstraction, by parameterization, or generated by explanation equivalence laws [**?**]. Additionally, languages for XOP should help guide the programmer toward the creation of *good* explanations.

The need for interactive explanation artifacts is motivated by the observation that there is a trade-off between personal explanations and traditional explanation artifacts, which can be partially bridged by XOP programs viewed as rich, interactive explanation artifacts. A good *personal explanation* is useful because the explainer can *respond* to the student, adjusting the pace and strategy as necessary. For example, the teacher can answer questions, rephrase parts of an explanation, and provide additional examples as needed. Unfortunately, good personal explanations are a scarce resource. First, there are limited number of people who can provide high quality personal explanations on a topic. Second, a personal explanation is usually ephemeral and so cannot be directly shared or reused. Since personal explanations are hard to come by, many students learn from *impersonal explanation artifacts*, such as recorded lectures, textbooks and online written nd graphical resources. These impersonal explanations lack the interaction and adaptability of personal explanations, but have the advantage of being easy to massively share and reuse via libraries and the internet.

In-person lectures, such as those covering algorithms in most undergraduate computer science programs, exist at a midway point between impersonal and personal explanations, perhaps closer to the personal end of the spectrum. These *classroom explanations* are adaptable—students can ask questions in class, the teacher can respond, and explanations can be adapted on the fly if students are confused—but they are not as adaptable as personal explanations since the teacher must accommodate many students at once. Classroom explanations are more efficient than personal explanations since they are shared amongst many students, but not as efficient as impersonal explanations since they are still ephemeral and therefore difficult to reuse.

We target another midway point, a bit closer to the impersonal end of the spectrum, of *interactive explanation artifacts* that provide as much of the responsiveness and adaptability of personal explanations as possible, but which can still be massively shared and reused online. Such an explanation artifact would be quite expensive to produce with current tools since an *explanation designer* must not only construct a high quality initial explanation and corresponding visualizations, but also anticipate and explicitly program in responses to queries by the student. We expected that DSLs for XOP can help alleviate this burden.

[conference]IEEEauthorblockA qtree

[.IP [.NP [.Det *the*] [.N [.N *package*]]] [.I [.I 3SG.PRES] [.VP [.V [.V *is*] [.AP [.Deg *really*] [.A [.A *simple*] *to use*.CP
]]]]]]