Designing Declarative Language Tutorials: A Guided and Individualized Approach

Anael Kuperwajs Cohen
Macalester College, St Paul, MN, USA
akuperwa@macalester.edu

Wode Ni
Institute for Software Research, Carnegie Mellon University, Pittsburgh, PA, USA
`http://www.cs.cmu.edu/~woden/`
woden@cs.cmu.edu

Joshua Sunshine
Institute for Software Research, Carnegie Mellon University, Pittsburgh, PA, USA
`https://www.cs.cmu.edu/~jssunshi/`
sunshine@cs.cmu.edu

Abstract

The ability to declare what a program should include rather than how these features should be implemented makes declarative languages very useful in many visual output programs. The wide-ranging uses of these programs, in domains ranging from architecture to web programming to data visualization, encourages us to find an effective method to teach them. Traditional tutorial systems are usually non-interactive and have a gap between the learning and application. This can leave the user frustrated without a way to move forward in the learning process. A general lack of guidance can lead the student down an incorrect path. To prevent these difficulties, we propose a guided tour followed by novel question types that both direct the student's learning and creates a focused environment to practice individual skills. Lastly, we propose a study to test the hypothesis that this tutorial is quicker to complete and results in a greater understanding of the declarative language.

2012 ACM Subject Classification Applied computing → Interactive learning environments

Keywords and phrases Declarative Programming, Programming Language Tutorial, Visualizations

Digital Object Identifier 10.4230/OASIcs.PLATEAU.2019.4

1 Introduction

Declarative languages have been successful in many domains because of the multiple advantages they possess. The readability [1], succinct composition, and unordered nature of the code can make them easier to use [6]. For creating programs with visual output, declarative languages are especially prevalent. There are many examples that are widely used, including HTML, CSS, D3, and others. The reason for that is the ability to declare what the aspects of the visualization should be rather than how they should be built [11]. Due to the common usage of declarative languages and their prominence with visualizations, we are investigating how to effectively teach a declarative language. We are conducting this investigation within the context of a mathematical diagramming and education system called PENROSE, which utilizes a declarative language, SUBSTANCE, that resembles standard mathematical notation. An uncomplicated, accessible way to learn SUBSTANCE would support the system overall. As a solution to our primary research question, we propose using a guided tour followed by a series of novel question designs that provide targeted, focused application practice.

A guided tour is a context-sensitive tutorial that uses constraints and checkpoints to guide the user through the different aspects of the program. Many video games use guided tours instead of multi-page manuals because there is less frustration among users [2]. Each time there is a new skill to be learned, the tutorial will show the player how to complete the action through instruction and practice, followed by a checkpoint. The tasks are smaller pieces of the overall game, so they build upon each other to complete the tutorial. Both video games and declarative languages consist of many small pieces and skills that build up into proficiency, therefore it is likely a guided tour will be an effective strategy for declarative languages. The possible downfalls that this method avoids, however, are seen with traditional programming language tutorials. The mixture of textual instructions followed by exercises leads to a gap between learning and applying. Removing this gap should result in faster learning [7].

To secure a smooth transition into using the system freely, there will be practice questions that follow the guided tour. These questions come in a set of novel designs that are more focused on honing individual skills. Typical models, on the other hand, have exercises that slowly build to an activity that encompasses all of the learned skills. This does not take into account the needs of the student and which areas require more practice. Providing targeted feedback through specific questions, with the appropriate level of difficulty, will ensure the student effectively learns all the necessary skills for using the declarative language [12].

Along with this design, having a visualization as an aid can be extremely useful in the field of education. Due to the undeniable connection between visual outputs and declarative languages, the educational benefits of diagrams are important as well. External representations, such as diagrams, portray the information in an alternative way and assist the learning process. Often, seeing the information visually, as opposed to textually, promotes better reasoning and problem solving [9]. Additionally, inferring information from visuals is easier due to their emergent properties [8]. When visualizations are paired accurately with a student's current capabilities and the task at hand, which is the aim of the practice problems, there are cognitive benefits that enhance learning [3].

In the following sections we introduce our ongoing efforts of designing a tutorial for declarative languages. We start by presenting the design of a guided tour that teaches the declarative language of PENROSE, SUBSTANCE. Following that, we will show the designs of the novel problem types that provide focused but unconstrained practice within the PENROSE system. Finally, we propose a user study that evaluates our design.

2 Tutorial Design

A successful tutorial ensures that the student can understand and utilize the content they just learned. Teaching a language, specifically, entails covering all aspects of the language that might be used afterwards. Programming languages often have formal descriptions of the grammar that it implements, such as the BNF. When looking at a declarative, domain-specific language, however, the size of the grammar is smaller, limiting the space of possible programs. Our system can leverage an explicit specification of the language constructs to automatically

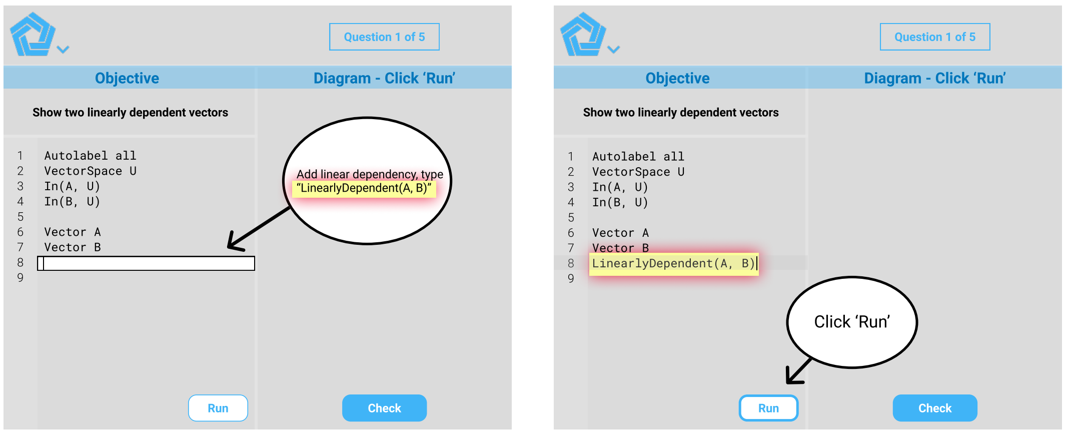
```
-- Type Constructors

type VectorSpace
type Vector

-- Predicates

predicate In: Vector * VectorSpace V
predicate LinearlyDependent: Vector * Vector
```

Figure 1 Domain Program.

generate instructional content such as guided tasks and practice problems. The system derives examples and counterexamples from the high-level grammar to create practice problems that target a specific language construct. In our example, PENROSE defines a mathematical domain with a DOMAIN program written in a simple metalanguage that declares all the types and operators available (see Figure 1). By sampling the space of possible SUBSTANCE programs defined by DOMAIN, the system generates various types of diagrammatic practice problems.

We have not yet generated problems from DOMAIN and there are many open research questions to solve to do so effectively. For example, when generating incorrect answers for multiple choice questions we will want to replicate plausible errors that students might make. Addressing these research questions is future work.

With our system, the guided tour consists of a set of tasks that covers different parts of the domain. Each task is broken up into a series of small steps. In order for the user to learn exactly what is intended, we guide the user through each step and constrain how the user can interact with the interface to ensure completion. We direct the user's attention towards a particular part of the question to show the next step via arrows, changes to the opacity of components in the user interface, and precise directions. Furthermore, the user cannot proceed without completing each step.



Show two linearly dependent vectors

```
Autolabel all
VectorSpace U
In(A, U)
In(B, U)

Vector A
Vector B

```

Add linear dependency, type "LinearlyDependent(A, B)"

Show two linearly dependent vectors

```
Autolabel all
VectorSpace U
In(A, U)
In(B, U)

Vector A
Vector B
LinearlyDependent(A, B)

```

Click 'Run'

(a) Guided Tour Step 3 (b) Guided Tour Step 4

Figure 2 Guided Tour.

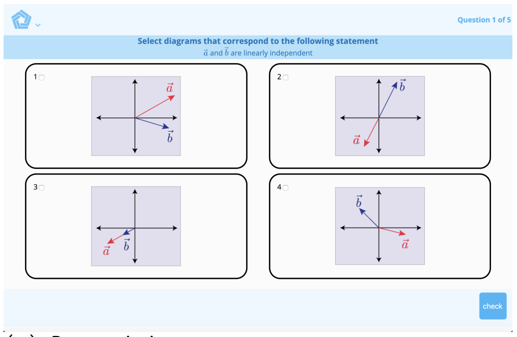
The purpose of using instructions at every step is to avoid frustration. Without personal assistance, advancing in the face of difficulties is a challenging task. In a system such as PENROSE, where the purpose is creating mathematical diagrams with ease, we aim to support users without any programming experience. Our goal in the guided tour is that the user should know exactly what they need to do at each step, and thereby increase retention rates [13]. At the same time, combining instruction with the physical task of writing code aids retention [2].

An example task for the guided tour (see Figure 2) has the prompt: "show two linearly dependent vectors", separated into smaller steps:

- Add the first vector, type `Vector A`
- Add a second vector, labeled `B`
- Add linear dependency, type `LinearlyDependent(A, B)`
- Click **Run**
- If the diagram looks linearly dependent, click **Check**

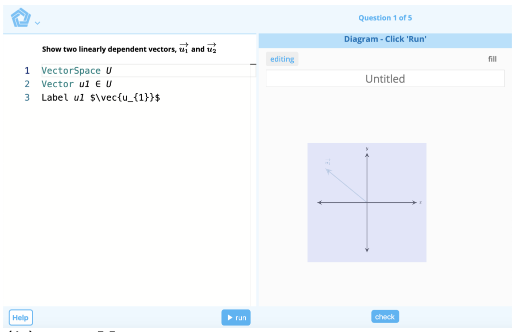
As seen in Figure 2, the opacity of the screen is lower, except for the instructions and the spot where the arrow is pointing. The instructions are contained within a boldly outlined ellipse, and the arrow points to where those instructions should be applied, which is also outlined. All these elements directs the attention of the user.

After progressing through the tasks, users naturally continue practicing with a set of novel question types. These questions utilize the PENROSE system, as described earlier in this section, and visualizations to emphasize the learning that occurs during the guided tour. The questions are focused on practicing specific skills and target the areas where the student is not as strong. The two categories of questions include recognition and recall.


Select diagrams that correspond to the following statement

$\vec{a}$ and $\vec{b}$ are linearly independent

(a) Recognition


Show two linearly dependent vectors, $\vec{u_1}$ and $\vec{u_2}$

```
VectorSpace U
Vector u1 ∈ U
Label u1 $\vec{u_{1}}$
```

(b) Recall

Figure 3 Novel Question Types.

Recognition is a category of problems that requires the student to recognize associations between the program outputs and the source programs. The methodology would be analyzing and working with the diagrams from the system. One type of problem that falls into this category is a multiple choice question where users match the code or prompt to the correct diagram (see Figure 3a). The prompt might ask for two linearly independent vectors, where two out of the four options are correct. Another question type is adjusting a diagram to match the code that is provided, meaning the user corrects the error. An extension of this problem is building the diagram from a bank of shapes.

Recall questions strengthen a student's ability to recall language constructs of the DSL. This involves a diagram that is already produced and constructing the answer. Our design supports this in two ways, correcting existing code and writing new code. For the former, there is an error in the code that the user has to find and fix. The latter is starting from the beginning and writing the code that answers the prompt (see Figure 3b). If the prompt requests two linearly dependent vectors, the user must write the code that creates that diagram.

3 Study Design

Implementing a study could provide evidence that supports this tutorial design. The purpose of an evaluation is to show the benefits of the previously described design, mainly that it is a faster and more efficient way of learning a declarative language. We hypothesize that this method will increase the understanding of the user.

Our tentative evaluation plan consists of students with a range of computer science backgrounds split into two groups, control and experimental. The control group learns SUBSTANCE with textual instructions, and have full access to the system. The experimental group uses the guided tour and the novel question types. The dependent measures are the time it took to complete the tutorial and the understanding, which is assessed through a quiz. The quiz will involve a combination of answering questions about example SUBSTANCE programs and writing SUBSTANCE code. This will be evaluated for accuracy. Every participant completes a pre and post-survey, followed by a debrief of the study.

4 Related Work

There are two main categories of related work that are correlated to our research question: visual learning and language tutorials.

Visual Learning

One focus of visual learning is how to improve visualizations to help students the most. The question's presentation has one of the largest impacts [3]. For example, effective instructions displayed alongside the external representation greatly increases understanding. Furthermore, the student must have enough experience to fully utilize the diagram. Grounded feedback is one way to use a student's prior knowledge [12], where they solve problems using a symbolic representation followed by the answer presented in a feedback representation. This second representation is a more concrete, familiar visualization and encourages students to interpret their answer's accuracy. Grounded feedback is a great tool for teaching, yet making the feedback individual is difficult to implement.

Visualizations can also be used as examples. Along with visual learning, example-based learning has become more common within computer science [5]. Unfortunately, visualizations and examples can be ineffective if they do not engage the student [10].

Language Tutorials

The most common way to learn a new language is through non-interactive methods, such as written text and videos, similar to Khan Academy.[1] Often, practice exercises and do-it-yourself tasks follow the explanations, leaving a gap where information can be forgotten. More interactive tutorials include websites such as Scrimba[2] and A Tour of Go,[3] that mix textual instructions and videos with practice questions. In Scrimba, students can interact with the code as the video is playing, making it easier to test out concepts as they are being taught. A Tour of Go is similar, but with textual instructions instead.

DrScheme [4] (now known as Racket) and the stencils-based tutorial for Alice [7] are examples of teaching within programming environments. DrScheme's purpose is to easily correct the errors many users run into. Experienced users have created work-arounds for these problems, but beginners get stuck. The stencils tutorial for Alice is similar to the guided tour, but focuses on teaching the system rather than a declarative language. This tutorial uses translucent stencils that direct attention to a hole that is regularly colored. Virtual sticky notes display the instructions. After comparing their new tutorial to a traditional one, they found that fewer errors were committed and it was a faster and easier experience overall. The main downside is that users who completed the stencils tutorial were less confident they could work with the program, even though they were more confident that they completed the tutorial correctly.

[1] `https://www.khanacademy.org`
[2] `https://scrimba.com`
[3] `https://tour.golang.org`

5 Conclusion

Since declarative languages are so widely utilized by users of all skill levels, teaching them efficiently is an important problem to investigate. Combining a guided tour and practice problems with new designs that focus on building individual skills suggests a more effective method. A future study could confirm this hypothesis, by looking for how fast the tutorial is completed and overall understanding of the content. This would improve the visual output systems that depend on declarative languages by providing more guidance for students without requiring personal assistance.

References

1 UW Interactive Data Lab | Papers. URL: `http://idl.cs.washington.edu/papers/d3/`.

2 Erik Andersen, Eleanor O'Rourke, Yun-En Liu, Rich Snider, Jeff Lowdermilk, David Truong, Seth Cooper, and Zoran Popovic. The impact of tutorials on games of varying complexity. In *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems - CHI '12*, page 59, Austin, Texas, USA, 2012. ACM Press. `doi:10.1145/2207676.2207687`.

3 Julie L. Booth and Kenneth R. Koedinger. Are diagrams always helpful tools? developmental and individual differences in the effect of presentation format on student problem solving. *The British Journal of Educational Psychology*, 82(Pt 3):492–511, September 2012. `doi:10.1111/j.2044-8279.2011.02041.x`.

4 Robert Bruce Findler, John Clements, Cormac Flanagan, Matthew Flatt, Shriram Krishnamurthi, Paul Steckler, and Matthias Felleisen. DrScheme: a programming environment for Scheme. *J. Funct. Program.*, 12:159–182, 2002. `doi:10.1017/S0956796801004208`.

5 Sumit Gulwani. Example-based Learning in Computer-aided STEM Education. *Commun. ACM*, 57(8):70–80, August 2014. `doi:10.1145/2634273`.

6 J Heer and M Bostock. Declarative Language Design for Interactive Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1149–1156, November 2010. `doi:10.1109/TVCG.2010.144`.

7 Caitlin Kelleher and Randy Pausch. Stencils-based tutorials: design and evaluation. In *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '05*, page 541, Portland, Oregon, USA, 2005. ACM Press. `doi:10.1145/1054972.1055047`.

8 Kenneth R. Koedinger. Emergent properties and structural constraints: Advantages of diagrammatic representations for reasoning and learning. In *Proc. AAAI Spring Symposium on Reasoning with Diagrammatic Representations*, pages 154–169, 1992. URL: `http://www.aaai.org/Papers/Symposia/Spring/1992/SS-92-02/SS92-02-031.pdf`.

9 Jill H. Larkin and Herbert A. Simon. Why a Diagram is (Sometimes) Worth Ten Thousand Words. *Cognitive Science*, 11(1):65–100, January 1987. `doi:10.1016/S0364-0213(87)80026-5`.

10 Thomas L Naps, Rudolf Fleischer, Myles McNally, and Alma College. Exploring the Role of Visualization and Engagement in Computer Science Education.

11 Arvind Satyanarayan, Kanit Wongsuphasawat, and Jeffrey Heer. Declarative Interaction Design for Data Visualization. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*, UIST '14, pages 669–678, New York, NY, USA, 2014. ACM. event-place: Honolulu, Hawaii, USA. `doi:10.1145/2642918.2647360`.

12 Eliane S. Wiese and Kenneth R. Koedinger. Designing Grounded Feedback: Criteria for Using Linked Representations to Support Learning of Abstract Symbols. *International Journal of Artificial Intelligence in Education*, 27(3):448–474, September 2017. `doi:10.1007/s40593-016-0133-9`.

13 A. Yan, M. J. Lee, and A. J. Ko. Predicting abandonment in online coding tutorials. In *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 191–199, October 2017. `doi:10.1109/VLHCC.2017.8103467`.