Towards a Theory of HtDP-based Program-Design Learning

Francisco Enrique Vicente G. Castro
Department of Computer Science
Worcester Polytechnic Institute
Worcester, Massachusetts, USA
fgcastro@cs.wpi.edu

ABSTRACT

Program-design is an essential skill students in introductory computing courses must learn, but which continues to be difficult for students. Many introductory curricula focuses on low-level constructs, even when students are expected to gain higher-level problem-solving and program-design skills. *How to Design Programs* (HTDP) is a curriculum that teaches a multi-step approach to program-design, promoting multiple, interrelated program-design skills. My research explores how novice programmers use HTDP-based techniques to design programs, the design-related skills students learn and use, the factors that drive their design decisions, and how these weave into a conceptual framework of HTDP-based program-design.

KEYWORDS

Program-design; CS1; novice programmers; qualitative research

1 PROGRAM CONTEXT

I am a fourth year PhD candidate in the Computer Science program at WPI; my program includes learning sciences courses to inform my research. I have defended my dissertation proposal in the early-Spring of 2018. My early research explores the planning behavior of CS1 students. I have built on this work through think-alouds and interviews with a new cohort of early-CS university students from which I have developed a SOLO-based framework of program-design-related skills and narratives of how students use the HTDP process to design programs. I will begin the next iteration of my studies in the upcoming school year.

2 CONTEXT AND MOTIVATION

Learning program-design remains a nontrivial goal for novice programmers in CS1 [2, 5, 12], requiring students to make various design choices: from lower-level concerns of choosing relevant programming language constructs to higher-level concerns of identifying and clustering subtasks into code blocks. Plan-composition is noted as a major difficulty among novices [12] yet most introductory programming courses focus heavily on teaching low-level programming constructs even when students are also expected to develop higher-level programming and problem-solving strategies, often through trial-and-error [5].

How to Design Programs [6] is an introductory computing curriculum that teaches a multi-step process of program-design. It has been adopted in higher-education institutions and some K-12 programs, yet the program-design skills it fosters and how students learn with HTDP remains largely unexplored in CSEd research. My work explores how novice programmers use the HTDP process to design programs, looking at the relationships around their use of program-design-related skills and techniques, and the contributing factors that drive their programming. Understanding how program-design-related skills and techniques are used, and the affordances and limitations around these provides valuable insight for designers of CS curricula and pedagogy.

3 BACKGROUND & RELATED WORK

More recent investigation into students' program-design skills have mostly looked at code outputs of students who are learning in different curricula [11] or after interventions that teach design strategies [5, 10]. These have showed varied results and often focused on *code-level techniques* (e.g. merging code). Others captured students' *mastery* of specific programming-related skills by assessing student output using taxonomies of skill progressions [8, 9].

On the other hand, HTDP [6] teaches students to work through a progression of steps when designing programs. Some of these steps include writing concrete examples of data, writing test cases for proposed functions, and writing code skeletons (*templates*) that fully traverse the input type. From this perspective, program-design isn't just a single strategy of merging relevant code blocks [9] or the application of recurring patterns [10], but also involves strategies such as using tests to model program behavior or designing programs based on data types instead of simply selecting language constructs. This multi-step process promotes the learning and use of techniques and multiple interrelated skills for program-design. This research aims to develop a more nuanced understanding of how students design programs using HTDP-based design techniques and skills and the factors that drive their design decisions.

4 STATEMENT OF THESIS/PROBLEM

The *How to Design Programs* curriculum teaches learners program-design through the development of a set of multiple, interrelated component skills [3]. We want to develop a conceptual framework of how novice programmers use HTDP to design programs. Interesting sub-questions include:

(1) What skills do HTDP-trained students display when they use HTDP to design programs? How might variations in the ways that students perform these skills look like?
(2) What affordances, difficulties, or limitations of the HTDP process can be observed from accounts of students' use of HTDP?
(3) How might differences in programming problem context influence students' use of HTDP? (e.g. solving problems for which students have seen applicable solution-structures vs. no prior knowledge on applicable solution-structures)

ACM ISBN 978-1-4503-5628-2/18/08.
https://doi.org/10.1145/3230977.3231020

(4) What other factors seem to influence students' (a) use of program-design-related skills and/or (b) use of HTDP-based design techniques and what relationships among these do we observe?

5 RESEARCH GOALS & METHODS

I approach my research questions with the following methods:

• **Conduct interviews and think-alouds with students.** I will conduct studies with students in HTDP-based CS1 courses that involve (1) giving the students programming problems to solve from scratch while thinking-out loud and (2) interviewing students about their approaches towards solving programming problems. These provide opportunities to engage with students and capture students' narratives about how they're using HTDP to design programs. The interviews will focus on the skills students are using (sub-question 1), accounts of their use of HTDP-based techniques (sub-questions 2, 3), and other factors and relationships that may influence their skill/technique-use and overall design practice (sub-question 4).

• **Develop a framework of program-design skills and narratives of students' use of HtDP.** We will code the think-aloud and interview data to identify relevant program-design skills. To date, we have developed a SOLO-based [1] framework that details the skills students use in their program-design practice, as well as the variations in the way each skill is applied. We will also code for accounts of how students use the techniques promoted by HTDP, with a particular focus on their use of the HTDP template design pattern. On top of these, we will also identify other underlying factors that may influence students' use of these program-design related skills and techniques. These analyses will enable us to gain an understanding of the skills that HTDP fosters, a sense of how to measure them, and a sense of the extent to which these skills and techniques can be applied – the affordances they offer, the difficulties that arise in their use, and their observed limitations.

• **Validate the conceptual frameworks of HtDP-use.** Developing the HTDP-based conceptual framework of novice program-design raises the subgoal of validating this framework to determine whether it accurately describes how novices use HTDP to design programs. We approach this validation in two ways: (1) we will replicate our studies and analyses across student cohorts from (1 to 2) other institutions that use an HTDP-based CS1 curriculum and (2) we will have other HTDP-expert instructors replicate our analyses on a subset of our data. The first approach allows us to generalize our findings across different HTDP student cohorts and account for differences in learning contexts (e.g. instructors, programming problems used in courses, etc.). The second allows us to check whether our framework captures similar nuances and relationships observed by other experienced HTDP instructors.

6 DISSERTATION STATUS

I have completed an exploratory study [2] of student programming behavior as they worked on novel problems. Findings show that students were unable to adapt their design processes to solve complex problems, tinkering on-the-fly rather than planning ahead. In a follow-up study [4], where we asked students to both produce and evaluate multiple structurally different solutions for a set of programming problems, students raised interesting factors that drove their design choices in evaluations of their own work. This led us to expand our work: I conducted think-alouds and interviews with students at multiple points during the course of their CS1 and through to CS2 to identify skills and influencing factors in students' design decisions and explore how these interact with their use of HTDP-based design techniques. Our analysis of the CS1 data resulted in the development of a SOLO-based framework [3] of the program-design-related skills and the variations in ways students perform these skills, as well as the development of narratives of students' use of HTDP-based design techniques to solve multi-task programming problems [7]. Going forward, I will validate and refine this framework by analyzing similar data from HTDP-trained students in 1 to 2 other institutions, as well as have other HTDP instructors replicate our analyses on a subset of our data.

7 EXPECTED CONTRIBUTIONS

My dissertation contributes to the development of CS education pedagogy. Understanding the meaningful distinctions in the variations of novices' skill-performance enables educators to assess instructional material to concretely identify content that support the development of particular design-related skills. Furthermore, understanding these skills and how students use design techniques enables us to capture the cognitive nuances that support the use of these skills and techniques. Second, it provides evidence towards the efficacy of an alternative method for program-design for novices. Understanding the affordances and accounts of how students use the HTDP-techniques in practice provides evidence towards the effectiveness and limitations of the techniques. In particular, the template design pattern in HTDP departs from traditional, implementation-focused patterns by drawing on data types instead of mappings between problem types and stereotype-patterns. This provides a useful alternative program-structuring method for novices who may not have experience or schema for certain problems, but can draw on data types to retrieve viable schemas.

REFERENCES

[1] J. B. Biggs and K. Collis. 1982. *Evaluating the Quality of Learning: the SOLO taxonomy.* Academic Press, New York.

[2] Francisco Enrique Vicente Castro and Kathi Fisler. 2016. On the Interplay Between Bottom-Up and Datatype-Driven Program Design. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education.* ACM, 205–210.

[3] Francisco Enrique Vicente Castro and Kathi Fisler. 2017. Designing a Multi-faceted SOLO Taxonomy to Track Program Design Skills Through an Entire Course *(Koli Calling '17).* ACM, New York, NY, USA, 10–19.

[4] Francisco Enrique Vicente Castro, Shriram Krishnamurthi, and Kathi Fisler. 2017. The Impact of a Single Lecture on Program Plans in First-year CS *(Koli Calling '17).* ACM, New York, NY, USA, 118–122.

[5] Michael de Raadt, Richard Watson, and Mark Toleman. 2009. Teaching and Assessing Programming Strategies Explicitly. In *Proceedings of the Eleventh Australasian Conference on Computing Education.* Darlinghurst, Australia, Australia.

[6] Matthias Felleisen, Robert Bruce Findler, Matthew Flatt, and Shriram Krishnamurthi. 2001. *How to Design Programs.* MIT Press. http://www.htdp.org/

[7] Kathi Fisler and Francisco Enrique Vicente Castro. 2017. Sometimes, Rainfall Accumulates: Talk-Alouds with Novice Functional Programmers *(ICER '17).* ACM, New York, NY, USA, 12–20.

[8] David Ginat and Eti Menashe. 2015. SOLO Taxonomy for Assessing Novices' Algorithmic Design *(SIGCSE '15).* ACM, New York, NY, USA, 452–457.

[9] Cruz Izu, Amali Weerasinghe, and Cheryl Pope. 2016. A Study of Code Design Skills in Novice Programmers Using the SOLO Taxonomy *(ICER '16).* ACM, 251–259.

[10] Orna Muller, David Ginat, and Bruria Haberman. 2007. Pattern- oriented Instruction and Its Influence on Problem Decomposition and Solution Construction *(ITiCSE '07).* ACM, New York, NY, USA, 151–155.

[11] Otto Seppälä, Petri Ihantola, Essi Isohanni, Juha Sorva, and Arto Vihavainen. 2015. Do We Know How Difficult the Rainfall Problem is? *(Koli Calling '15).* ACM, 87–96.

[12] E. Soloway. 1986. Learning to Program = Learning to Construct Mechanisms and Explanations. *Commun. ACM* 29, 9 (Sept. 1986), 850–858.