An Instructional Model to Link Designing and Conceptual Understanding in Secondary Computer Science Education

Ebrahim Rahimi
Radboud University &
Open University
ebrahim@cs.ru.nl

Erik Barendsen
Radboud University &
Open University
e.barendsen@cs.ru.nl

Ineke Henze
Delft University of Technology
f.a.henze-rietveld@tudelft.nl

ABSTRACT

In design-based education, students use scientific concepts to inform their artifact-making endeavors. On the other hand, artifact-making activities are meant to deepen students' conceptual understanding. However, no strategy has been described that explicitly links conceptual development and artifact-making endeavors in computer science design projects in an authentic way. To fill this gap, in this 'work in progress' study, we developed an instructional model for fusion designing and conceptual learning in CS education. A key component of the model is the 'intermediate design products' which play a critical role in connecting designing and conceptual learning. The model served to develop exemplary lesson materials meant to enhance students' algorithmic thinking. The materials were implemented and evaluated in four classes. The results suggest the model provides opportunities to improve students' algorithmic thinking. Furthermore, intermediate products turn out to be promising inputs for (formative) assessment of conceptual development in design projects by capturing and revealing students' misconceptions about basic CS concepts.

KEYWORDS

Design-based education, Computer science education, learning by designing, algorithmic thinking, conceptual development

ACM Reference Format:
Ebrahim Rahimi, Erik Barendsen, and Ineke Henze. 2018. An Instructional Model to Link Designing and Conceptual Understanding in Secondary Computer Science Education. In *Proceedings of the 13th Workshop in Primary and Secondary Computing Education (WiPSCE '18), October 4–6, 2018, Potsdam, Germany*. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3265757.3265768

1 INTRODUCTION

Design-based education is a popular instructional approach to teaching Computer Science (CS) in secondary schools. In this approach, students participate in individual or group projects for developing various types of digital artifacts including software applications, websites, games, and videos [2, 12]. There is an increasing tendency in design-based education toward realistic, ill-structured design problems with ill-defined goals, specifications, states, or operators [8, 9]. In the context of CS design-based education this tendency is boosted by professional software design processes characterized by facing with ill-defined problems; dealing with complex and often conflicting constraints; and generating large, complex, dynamic, and intangible artifacts [8, 16].

Computing has been advocated as an engineering as well as science-oriented discipline [3, 15]. However, most of the instructional models suggested for CS design projects in secondary education mainly address the engineering and development aspects of computing. In these models, the link between students' designing and conceptual development is not explicit and there is no mechanism to facilitate and reveal students' reasoning about their design and trace back their design features to their underlying concepts. To the best of our knowledge, there is no documented instructional strategy to make students' learning visible and deepen and assess their conceptual development in an authentic way (i.e., without disturbing the design context).

This study is part of an ongoing research project aiming at facilitating formative assessment of conceptual development in secondary school CS design projects. In this study, we developed an instructional model to introduce and use intermediate design products for fusion designing and conceptual learning in CS education. The model was used to inform the development of course materials to enhance students' algorithmic thinking through developing a digital text analyzer tool.

2 THEORETICAL BACKGROUND

The theory of constructionism forms the theoretical basis for this research. *Learning-by-making* forms the core of the constructionism theory [10]. From the perspective of this theory, learning is the conscious process of building knowledge structures through constructing an artifact whether "a sand castle on the beach or a theory of the universe" [10] (p. 1). Constructionism states that learning takes place when learners engage in design challenges to develop a "public entity" or a personally meaningful artifact and "audience to share insights with" while working on their designs [10]. However, some people warn that mere focus on activity-based learning experiences can lead to a form of "shallow constructionism" or building for the sake of doing where students make an artifact without gaining a deep understanding of its underpinning concepts [11, 14].

To fusion learning and design and prohibit shallow constructionism issues, Kolodner et al. [9] introduced the Learning-By-Design (LBD) framework. This framework aims to shape an educational approach to learning science concepts via working on real design

Permission to make digital or hard copies of all or part 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 components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
WiPSCE '18, October 4–6, 2018, Potsdam, Germany
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-6588-8/18/10…$15.00
https://doi.org/10.1145/3265757.3265768

challenges in parallel with acquiring cognitive, social, and communication skills in the context of middle schools [9]. To this end, LBD defines a process consisting of two interconnected and iterative "Design & Redesign" and "Investigate & Explore" cycles. This process starts with the "Design/Redesign" cycle seeking to understand, analyze and address the design challenge. When a specific scientific investigation or knowledge is needed (i.e., the *"Need to Know"* moment) the process switches to the "Investigate & Explore" cycle to test a hypothesis or learn the required concept. Then, the process returns back to the "Design/Redesign" cycle to apply or test the acquired findings (i.e. the *"Need to Do"* moment).

3 THE INSTRUCTIONAL MODEL

We adapted the Learning-By-Design (LBD) framework [9] for developing an instructional model to link conceptual learning and designing in the context of secondary CS education, as depicted in Fig. 1. The model consists of five interconnected components: *Design-oriented objectives*, *Knowledge-oriented objectives*, *Making cycle*, *Learning cycle* and *Intermediate design products*.

The *making cycle* is the core component of the model. It is informed by real and meaningful design objectives and its ultimate goal is to enhance students' algorithmic thinking through developing an IT artifact. To support algorithmic thinking, the making cycle provides four phases of analysis, design, development, and test [4]. These phases provide a general roadmap for students to guide and orchestrate their artifact-making activities and address the development aspect of computing as an engineering and design discipline [3]. Each phase requires and asks students to gain specific sorts of knowledge. The specifications of these phases are as follows:

To accomplish their design assignmentms students might need to gain specific knowledge and concepts including problem-solving, basic CS concepts, programming, and evaluation. To this end, they are encouraged to engage in various knowledge acquisition activities including reading content, performing unplugged activities, generating trace tables, answering questions, presenting and discussing returns, and completing or debugging a program. The knowledge acquisition activities might get evoked within the phases of the *making cycle* when more knowledge is required to accomplish the design assignment (the *Need to Know* moment as mentioned above). For example, in the test phase, an observed inconsistency between the program outputs and the design objectives might initiate a series of learning activities to acquire required knowledge to address this inconsistency. This represents a *just-in-time* approach to learning CS concepts which is well aligned to this fact that people naturally do not learn something just for the sake of learning but to do something meaningful and relevant with it [10]. This is contrary to the most traditional approaches to providing CS design-based education where students first learn CS concepts separately and then try to apply these concepts in practice. The knowledge acquisition part of the instructional model seeks to facilitate students' understanding of CS concepts and their implications and address the science-oriented aspect of computing [1, 15].

The joint outputs of the *making* and learning cycles are manifested in tangible student-generated intermediate design products, namely, *a conceptual solution* (i.e. a high level overview of the design

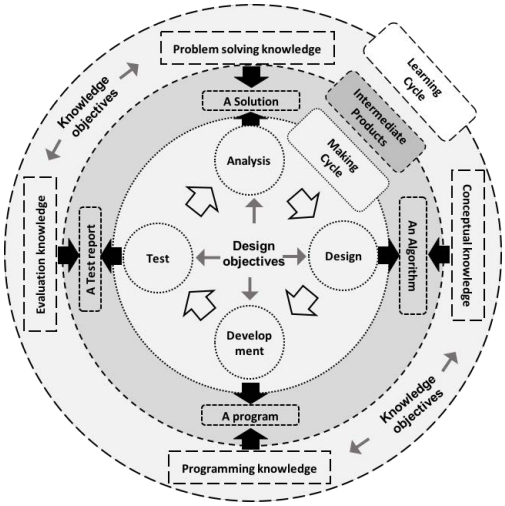

Figure 1: An instructional model for linking learning and design in the context of secondary CS education

problem including its inputs, outputs, and purposes; its parts and their interconnections; and an abstract formulation of an initial solution for it), *an algorithm* in terms of a flowchart representing a language-independent, visual, detailed and step-wise implementation of the solution, a *program*, and a *test report*. These intermediate products serve as evidence of students' learning and design outcomes resulted from their collaboration around the design challenge. They are meant to make students' learning, reasoning and decisions about design explicit. Since these intermediate products emerge during and from the design and learning processes, they seem promising inputs for formative assessment strategies.

4 THE STUDY SETTING

We used the instructional model to develop course materials to facilitate students' learning of basic algorithm concepts via extending a digital text analyzer tool. Then we conducted a research to investigate the benefits of the instructional model, in particular the intermediate design products directed by the following questions:

In the participants' perception, how do the intermediate design products:

(i) facilitate students' algorithmic thinking and reasoning about design?

(ii) are beneficial to students' understanding of the intended CS concepts?

The Developed Course Materials

We used the proposed model to generate course materials for developing a digital text analyzer tool (i.e. the *design objective*) and learning basic algorithm concepts (i.e. the *knowledge objectives*) by upper secondary school students. The course materials include two design and content documents. The design document included two examples and five real assignments. In each assignment students were asked to follow the making cycle and generate and share intermediate design products. To facilitate this process, we developed a group of sub-algorithms with intuitive names to be

used by students to synthesize their algorithms. In the design document an iterative approach built upon the completion programming strategy [17] was taken. The content document covered various aspects of algorithms including its basic concepts, trace table, and flowcharts. In addition to these documents, a software tool (for both PHP and Python programming languages) was developed to facilitate students' programming of their algorithms.

Participants

Four CS teachers and 87 students aged 15-16 from 4 secondary schools in the Netherlands participated in this project which lasted 8 weeks. All of the students had programming experience with either PHP or Python from their previous CS courses. However, none of them had prior knowledge and experience about algorithms.

Data Collection

In this in-depth and exploratory study, we opted to use interviews as the data collection method. The interview questions asked for participants' perceptions about the benefits of the project for enhancing their algorithmic thinking and their encountered difficulties. Five group interviews with sixteen students and four interviews with the teachers were conducted. Further, we collected students' generated solutions, flowcharts, and programs as secondary data to support the analysis.

Data Analysis

We categorized the benefits of the project for enhancing students' learning of algorithm concepts in four groups, namely, *problem analysis*, *devising algorithm*, *program development*, and *conceptual understanding* adopted from [6]. Within each category, the interview data were analyzed inductively.

5 RESULTS

We present the results according to the above categories. Within each category, we discuss the themes emerging from the inductive analysis.

Facilitation of Students' Algorithmic Thinking

Problem Analysis

Problem decomposition: According to the participants, decomposing a problem into sub-problems was quite new for them and they found it very interesting and useful.

Dividing a problem into subproblems and finding their interconnections was not an easy and interesting task for some of the students. Furthermore, some of the students with more programming knowledge and experience preferred to jump quickly to the programming part.

Splitting the design process: The interviewed students and teachers found the defined phases, namely, analysis, design, development, and test and their associated products a useful strategy to chunk the whole software development process and get an overall overview of it.

Analysis of possible inputs: The open, ill-defined and realistic nature of the design assignments faced the participants with complex challenges and also provided great opportunities for problem analysis, analysis of possible inputs for the problem, decision making and reflection.

Formulation of general solution: was perceived beneficial in translating a vague perception of the problem existing in someone's mind into an explicit and communicable picture of the problem and its components. This picture acts a road map for providing a general and abstract overview of the problem, its inputs, components and the required steps to solve it without getting confused by details.

Devising algorithm

Promoting abstract thinking: The model appeared to have promoted abstract thinking among the students through generating an abstract solution for their design assignments and using the provided sub-algorithms.

Triggering thinking ahead: The interviewed students enumerated several benefits for using flowcharts including working on smaller problems, achieving a high level of abstraction, promoting systematic and logical thinking, forcing students to think ahead, and giving more structure and organization to the solution and program.

Resolving group task-division issues: Some teachers found sub-algorithms as useful means to address task division issues within groups.

Increasing the traceability and communicability of solutions: Dividing a solution into sub-algorithms was perceived highly beneficial in making the solution more traceable and communicable for team members.

Dealing with uncertainty: The student-centric approach of the project, existing several answers for a problem and lack of access to correct answers faced students with challenges and at the same time provided them great opportunities to encounter and handle several uncertainties in their design decisions.

Despite these positive reactions, the participants encountered several difficulties including calling sub-algorithms properly in flowcharts, managing interaction between sub-algorithms and the main algorithm, and passing parameters. Also, generating flowcharts using pen and paper was perceived difficult and time-consuming by some of the interviewees.

Program development

Practicing programming competencies: Most of the groups completed their assigned assignments and produced working programs.

Using functions to organize and structure code: Students were asked to use already implemented sub-algorithms and their associated functions to implement their solutions and write their programs. Some of the interviewed students found these functions beneficial, in particular, for enhancing their abstract thinking, getting rid of writing a large number lines of code, and improving the quality and readability of their programs. One group even used and adapted the provided functions to define a new assignment and build it. On the other hand, several students used built-in functions of PHP or Python to develop their programs.

Benefits for Students' Conceptual Development

Making learning more relevant: Most of the interviewees remarked that the authentic and contextualized nature of the design

assignments promoted a *doing-then-learning* approach to learning CS concepts. According to them, this approach served to encourage their learning activities via spotting their knowledge gap and defining more realistic and meaningful purposes for their learning and making processes.

Shifting from 'did-you-know' to 'how-to-use' approach: Another perceived benefit of the project concerns with its contribution in changing the focus of learning from *did-you-know* to *how-to-use*.

Making learning visible using flowcharts: The intermediate design products, in particular, flowcharts were appreciated by most of the interviewees as useful means to make students' learning and misconceptions visible.

Improved learning curve: The completion-based approach followed by the instructional model was perceived useful for the gradual enhancement of students' learning and design skills.

6 DISCUSSION

The realistic and contextualized aspects of the design assignments appeared to be highly influential in motivating students' making and learning processes. There is the general agreement that conducting realistic projectscan promote deep and effective learning through situating learning in purposeful and engaging activities, connecting students to a context outside of the school boundaries and giving more professional sense to their endeavors rather than mere educational purposes [5, 9].

The instructional model appears to provide opportunities to facilitate students' algorithmic thinking through facing them with open and complex assignments where their analytical thinking was triggered to consider possible inputs for the design assignments and develop a solution to process those inputs. Such design assignments involved ambiguity and uncertainty where students felt unsure about the correctness of their answers. Dealing and coping with complexity and uncertainty are key competencies in computing [7].

The intermediate design products served to facilitate students' algorithmic thinking via several aspects: First, flowcharts were used as cognitive means to translate students' internal and vague thinking into explicit and step-wise solutions. Doing such, flowcharts served to facilitate students' sketching of the solution which was proposed as a deep cognitive process [7]. Secondly, student-generated flowcharts are likely to facilitate students' social interactions by providing opportunities for presenting ideas, collaboration, communication, sharing and exchanging their design knowledge, reasoning and decisions, and peer assessment. Thirdly, student-generated design products seem useful means for documenting, organizing, and tracing back the design decisions and reasoning of students by generating snapshots of their design process. These products can arguably provide explicit evidence of students' reasoning at three levels, namely, *why* (represented by their general solution or the abstract formulation of the problem), *how* (represented by their step-wise flowcharts), and *what* (represented by their generated program and its outputs).

The instructional model appeared to be beneficial to students' conceptual development in several ways: First, by using CS concepts practically and putting them together to build more complex constructs such as flowcharts or working programs. However, to understand more complex concepts such as sub-algorithms and nested loops students require more support from their teachers. Secondly, using intermediate design products makes students' learning visible. This proposes these products as suitable means to implement formative and peer assessment strategies [13].

7 CONCLUSIONS

In this study an instructional model was developed to link design and conceptual development in secondary school CS education. The intermediate design products can be seen as evidence of their reasoning and design decisions at three levels of why, how, and what. The model seemed to have contributed to students' conceptual development. Some evidence of 'shallow constructionism' were observed. This study will be completed by further implementation of the materials and various sorts of results including student-generated intermediate products to support the analysis.

REFERENCES

[1] Michael E Atwood, Robin Jeffries, Althea A Turner, and Peter G Polson. 1980. *The Processes Involved in Designing Software.* Technical Report. SCIENCE APPLICATIONS INC ENGLEWOOD CO.

[2] Erik Barendsen, Linda Mannila, B. Demo, N. Grgurina, C. Izu, C. Mirolo, S. Sentance, A. Settle, and G. Stupurienė. 2015. Concepts in K–9 Computer Science Education. In *Proceedings of the 2015 ITiCSE on Working Group Reports*. ACM, 85–116.

[3] Anders Berglund and Raymond Lister. 2010. Introductory programming and the didactic triangle. In *Proceedings of the Twelfth Australasian Conference on Computing Education – Volume 103*. Australian Computer Society, Inc., 35–44.

[4] Barry W. Boehm. 1988. A spiral model of software development and enhancement. *Computer* 21, 5 (1988), 61–72.

[5] Allan Collins, John Seely Brown, and Susan E Newman. 1989. Cognitive apprenticeship: Teaching the crafts of reading, writing, and mathematics. *Knowing, learning, and instruction: Essays in honor of Robert Glaser* 18 (1989), 32–42.

[6] Jill Denner, Linda Werner, and Eloy Ortiz. 2012. Computer games created by middle school girls: Can they be used to measure understanding of computer science concepts? *Computers & Education* 58, 1 (2012), 240–249.

[7] Clive L Dym, Alice M Agogino, Ozgur Eris, Daniel D Frey, and Larry J Leifer. 2005. Engineering design thinking, teaching, and learning. *Journal of Engineering Education* 94, 1 (2005), 103–120.

[8] Vinod Goel and Peter Pirolli. 1992. The structure of design problem spaces. *Cognitive science* 16, 3 (1992), 395–429.

[9] Janet L Kolodner, Paul J Camp, David Crismond, Barbara Fasse, Jackie Gray, Jennifer Holbrook, Sadhana Puntambekar, and Mike Ryan. 2003. Problem-based learning meets case-based reasoning in the middle-school science classroom: Putting learning by design (tm) into practice. *The journal of the learning sciences* 12, 4 (2003), 495–547.

[10] Seymour Papert and Idit Harel. 1991. Situating constructionism. *Constructionism* 36 (1991), 1–11.

[11] Ebrahim Rahimi. 2015. *A Design Framework for Personal Learning Environments*. Ph.D. Dissertation. Delft University of Technology, The Netherlands.

[12] Ebrahim Rahimi, Erik Barendsen, and Ineke Henze. 2016. Typifying Informatics Teachers' PCK of Designing Digital Artefacts in Dutch Upper Secondary Education. In *International Conference on Informatics in Schools: Situation, Evolution, and Perspectives*. Springer, 65–77.

[13] Ebrahim Rahimi, Erik Barendsen, and Ineke Henze. 2017. Identifying Students' Misconceptions on Basic Algorithmic Concepts Through Flowchart Analysis. In *International Conference on Informatics in Schools: Situation, Evolution, and Perspectives*. Springer, 155–168.

[14] M Scardamalia and C Bereiter. 2006. Knowledge building: theory, pedagogy, and technology (pp. 97-119).

[15] Matti Tedre. 2014. *The science of computing: shaping a discipline.* CRC Press.

[16] Josh D Tenenberg, Sally Fincher, Ken Blaha, Dennis Bouvier, Tzu-Yi Chen, Donald Chinn, Stephen Cooper, Anna Eckerdal, Hubert Johnson, Robert McCartney, et al. 2005. Students Designing Software: a Multi-National, Multi-Institutional Study. *Informatics in Education* 4, 1 (2005), 143–162.

[17] Jeroen JG Van Merriënboer. 1990. Strategies for programming instruction in high school: Program completion vs. program generation. *Journal of educational computing research* 6, 3 (1990), 265–285.