10-2017

Implementation Projects in a Computing Theory Course

Tammy VanDeGrift
University of Portland, vandegri@up.edu

IMPLEMENTATION PROJECTS IN A COMPUTING THEORY COURSE

Tammy VanDeGrift
Computer Science, Shiley School of Engineering
University of Portland
Portland, OR 97203
503-943-7256
vandegri@up.edu

ABSTRACT

Most computer science programs expose students to theoretical aspects of computing, such as discrete mathematics, algorithms, and theory of computation. This paper presents the integration of an implementation project in a theory of computation course, so that students get a chance to grapple with the details of a transformation and/or abstract model in addition to preparing a project and demonstration to help fellow students review topics from the course. Examples of student projects include a deterministic finite automata simulator, determining if the languages of two DFAs are equal, converting a grammar to a pushdown automaton, and creating a regular expression engine. Seventeen of 25 respondents agreed or strongly agreed that the project was a valuable learning experience; six were neutral and one strongly disagreed. Student project topics were reviewed against final exam questions for corresponding language classes. While there was no statistically significant difference between groups on exam questions, the overall averages on exam questions demonstrate student mastery of the material.

INTRODUCTION

Most computer science programs include theoretical aspects of computing, with courses such as discrete mathematics, theory of computation, and algorithms. In particular, theory of computation is the study of the capabilities and limitations of computers [7, 8, 11]. Computer science students who study theory of computation typically learn about formal computation using automata, such as deterministic finite automata and pushdown automata, Turing machines, and how to classify problems as decidable or undecidable. ABET-accredited programs include outcomes related to theoretical aspects of computing: (a) An ability to apply knowledge of computing and mathematics appropriate to the program's student outcomes and to the discipline and (j) An ability to apply mathematical foundations, algorithmic principles, and computer science theory in the modeling and design of computer-based systems in a way that demonstrates comprehension of the tradeoffs involved in design choices. [1].

Traditional homework exercises in a theory of computation course include proof-based problems, such as those provided in [7], [8], and [11]. For example, a homework exercise may ask students to show that the language consisting of strings that start and end with the same character is regular. Another exercise may ask students to show that the intersection of a context-free language and regular language is context-free. Yet another may ask students to prove that strings

consisting of strings in the form $a^N b^N c^N$ is decidable. These are important proof-creation skills for students to develop.

This paper provides an overview and assessment of a theory of computation course that includes a programming project in addition to traditional proof-based exercises. The goals of the programming project are to: 1) apply prior experience in programming to theoretical topics, 2) design, implement, and test a system (no starter code), and 3) communicate the project to others.

The constructivism learning theory states that people are active creators of knowledge – each person must grapple with ideas, explore, and integrate new ideas within the context of their own frameworks [2, 3, 6, 13]. Instructors aid students to be active learners by scaffolding student inquiry activities. By asking students to implement a simulator or transformation as a course project, they must grapple with the details and explore how to design, implement, and test their ideas.

Others have studied pedagogical techniques in theoretical computing courses. For example, Coffey utilized programming projects to experimentally verify runtimes in a data structures and algorithms course [4]. Walker has used oral presentations and oral final exams to assess students in a Theory of Computation course [14]. A popular way to demonstrate how strings are processed by automata is to use tools, such as JFLAP and other simulators [5, 10]. Liu also uses software project demonstrations for learning in addition to assessment [9].

CONTEXT

The Theory of Computation (TOC) course is required for the BSCS degree at a private University on the west coast of the USA. Prior to taking TOC, students must pass both discrete mathematics and data structures with a C- or better. TOC serves as a prerequisite for Compiler Design, also a required course in the curriculum. Typical section sizes for TOC are 30 – 35 students. Students are expected to learn how to construct proofs about language classification (see Table 1 for list of topics). For example, homework and exams ask students to prove a certain language is regular (or context-free, or decidable) and to also construct a proof to show a certain language is not regular (or not context-free, or not decidable). During the 15-week semester, students complete nine written homework assignments, small pieces of python code to demonstrate the construction of regular expressions, three midterm exams, a final exam, an art project [12], and a programming project. This paper focuses on the programming project.

Programming Project. The project is assigned five weeks into the semester after students have seen regular languages and context-free languages and will soon be starting to learn about decidable languages. Table 1 shows the schedule of the project as it relates to course topics. Table 2 in the results shows the projects that students completed. The project assignment is as follows:

> In order to practice what you have learned in CSXXX, you will complete a programming project that implements one of the theorems, transformations, or algorithms that we have seen in class. This project is worth 10% of the overall grade in the course. The demo of the project is worth 5% of the overall grade in the course. The project can be completed individually or in pairs. *If done in pairs, the scope of the project should be appropriate for two people.*

Table 1: Course Schedule and Project Deliverables

Week	*Topics*	*Project Information*
1	State Machines; DFAs	
2	NFAs; DFA <-> NFA	
3	Regular expressions; Pumping Lemma (Reg)	
4	Grammars; CFLs	
5	PDAs; Grammar <-> PDA; Pumping Lemma (CFL)	Project Assigned
6	Pumping Lemma; Turing Machines	
7	Turing Machines; Algorithms; Decidability	
8	Decidability; Halting Problem	Project Proposal Due
9	Undecidability	Project Approved
10	Undecidability; Rice's Theorem	
11	Post's Correspondence; Complexity; P	
12	NP, SAT, NP-completeness	
13	NP-completeness	
14	Demos; Review	Project Due; Project Demos
15	Final exam	

Students (or pairs) create a proposal about their project and submit it during the eighth week of the semester. Because one goal is to have the project demonstrations help other students review material for the final exam, the instructor limits duplicate projects to three per semester (or two for smaller sections). Student projects are approved first-come-first-served, so students who submit early have a better chance of their project being approved. In the project proposal, students must include name(s), a description of the theorem or transformation, the expected input(s) to the program (such a files or command line input), the expected output(s) of the program, the intended programming language(s), block diagram of the design, data structures, test cases for the program (can be hand-drawn), and (for pairs) responsibilities for each member of the pair.

The projects are due at the beginning of the last week of class. Students submit the code, test files, and a project summary. The summary includes name(s), documentation about how to compile/run the program, overall design of the project (diagrams, data structures, how data is encoded), testing results (include examples and limitations), and (for pairs) how each person contributed to overall project.

During the last week of the semester, each pair/individual is given five minutes to demonstrate the project. Students are allowed to create videos if doing a live demonstration would take too long or if computing systems cannot be brought into the classroom (such as a desktop computer). The instructor organizes the project topics to align with the order in which they were introduced during the semester. The five-minute time limit is imposed to ensure that all projects can be covered in the class sessions and to get students to focus on the essentials of the project: what is the topic, how the transformation/algorithm works in general and how it was designed/implemented, and a demo of the code execution. Student demos are graded according to organization, delivery, project design and implementation, and visual aids.

METHODS

The following questions were explored in this study:

- RQ1: How do students value the project in the context of learning?

- RQ2: What types of projects do students implement?
- RQ3: How does project topic relate to performance on related exam questions?

In order to answer RQ1, students from fall 2016 were asked the following questions at the end of the semester:

- The project was a valuable learning experience. (Strongly Agree, Agree, Neutral, Disagree, Strongly Disagree)
- How did the project help you learn? (open-ended text)

Projects were collected and categorized by topic for the past three offerings of the course to answer RQ2. Final exam grades were collected for questions about regular languages, context-free languages, and decidable languages. Exam scores from questions about regular languages were compared between students that completed a project related to regular languages and those that did not. The same process was used for context-free languages and decidable languages.

RESULTS

RQ1. 25 of 32 students from fall 2012 completed the survey in 2016. Figure 1 shows the results for the Likert-like question about the project, given as percentage of student respondents.

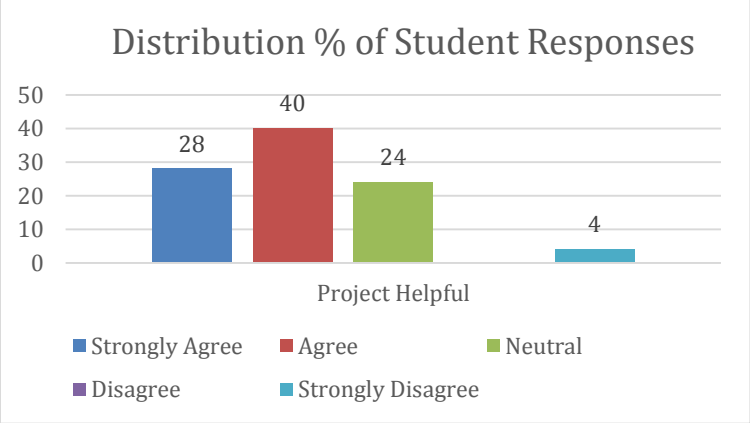

Figure 1: Responses for "The project was a valuable learning experience."

Twenty-four students provided responses to the open-ended question about how the project helped them learn. The responses were coded into emergent themes. 17 responses fit the theme of helping them apply concept from the course and 4 mentioned the real-life/real-world application of a theoretical concept. For example, one student said, "I learned how the formal descriptions relate to the diagram versions of DFAs/NFAs/PDAs/TMs." Another said, "I guess it made me feel like the stuff that we did is actually tangible. Like I can actually use them in real life which is hard to believe giving the amount of abstract stuff that we did." Other students commented more broadly about project implementation skills – 2 commented on the difficulty of the project, 2 commented on the ease of the project, and 2 commented on learning more about coding and software design. The last category consists of more general learning outcomes – 1 stated that it helped with creativity and 1 stated that it helped improve teamwork skills.

RQ2. Table 2 shows the project descriptions and number of students/pairs who completed that project for each semester. Between Fall 2015 and Spring 2016, the instructor added more examples of decidable languages involving DFAs to the project handout. For the most part, there was good coverage of regular languages, CFLs, and decidable languages for project topics. Three students (one individual and one pair) did projects related to the course but not on topics explicitly covered. The order in Table 2 mimics the order or project demos.

Table 2: Project topics and number of student/pairs completing the project

Project	*Fall 2015*	*Spring 2016*	*Fall 2016*
DFA Simulator	2	1	1
DFA Animator	1	2	1
NFA Simulator	1		
NFA Animator		2	
Union of Regular Languages	1	3	2
A* is Regular	2	1	2
Concatenation of Regular Languages	1	2	2
Intersection of Regular Languages			1
NFA -> DFA Conversion	1	1	2
DFA -> Regex Conversion	1	2	
Regex -> NFA Conversion	2	3	
Regular Expression Engine	1	1	1
PDA Simulator	2	1	1
CFG -> Chomsky Conversion	2	3	2
CFG -> PDA Conversion	3	1	
PDA -> CFG Conversion		1	2
Deterministic TM Simulator	1	2	2
L(DFA) is infinite is Decidable			2
L(DFA) is empty is Decidable		1	2
L(DFA1) = L(DFA2) is Decidable		1	2
Password Generator		1	
Graph Visualizer			1

RQ3. Over the three semesters, 99 students completed projects and the final exam. Each student was categorized into one of four groups related to the project type they completed: Regular (R), Context-Free (C), Decidable (D), Other (O). Exam scores for questions related to each language class were compared between groups. Table 3 shows the results. None of the p-values are statistically significant for a one-way ANOVA; student project completion did not influence final exam performance on questions related to that language class. Students saw all project demos the last week of class, so perhaps all students benefited from this review of regular languages, CFLs, and decidable languages. It is also the case that topics associated with the projects were presented in the first seven weeks of the semester, so there was plenty of time for students to practice and review this material.

Table 3: Data Comparing Final Exam Scores Between Groups (R, C, D)

	Average Score	*Max Score*	*Std Dev*	*p-value*
R (N=57)	24.33	28	2.77	.7135
Non-R (N=42)	24.12	28	2.98	

C (N=24)	24.06	28	3.93	.8597
Non-C (N=75)	24.22	28	3.74	
D (N=15)	17.40	20	2.58	.3171
Non-D (N=84)	16.36	20	3.86	

CONCLUSIONS

This study shows that it is possible to combine theory and practice through the use of programming projects. Not only did students get to implement a specific transformation/simulation, they presented what they did as part of a week-long review session in the course. All but one student was neutral or positive about the effectiveness of the projects in terms of helping them learn. Students' perceived benefits include applying concepts from the course, general programming practice with design and implementation, application of creativity, and learning teamwork skills. Even though there were no statistically significant differences on final exam questions, the overall scores on the final exam questions demonstrate learning of theoretical material. The average exam scores on the project-related material ranged from 82.5 to 86.5%. In summary, students valued the project in terms of their learning, the projects provided value to the entire class for review, and the students demonstrated mastery of the material.

REFERENCES

[1] ABET, www.abet.org, last accessed May 25, 2017.

[2] Bransford, J., Brown A.L., Cocking, R.R. *How People Learn: Brain, Mind, Experience, and School*, National Academies Press, 2000.

[3] Bruner, J.S., The act of discovery, *Harvard Educational Review*, 31 (1), 21-32, 1961.

[4] Coffey, J.W., Integrating Theoretical and Empirical Computer Science in a Data Structures Course, *Proceedings of the ACM SIGCSE Symposium*, 2013.

[5] Chuda, D., Rodina, D., Automata Simulator, In *Proceedings of the International Conference on Computer Systems and Technologies*, 2010.

[6] East, J.P., On Models of and for Teaching: Toward Theory-Based Computing Education, *Proceedings of ACM International Computing Education Research* Workshop, 2006.

[7] Hopcroft, J. E., Motwani, R., Ullman, J. D., *Introduction to Automata Theory, Languages, and Computation*, third edition, Pearson, 2006.

[8] Lewis, H. R., Papadimitriou, C. H., *Elements of the Theory of Computation*, second edition, Prentice-Hall, 1997.

[9] Liu, C., Software Project Demonstrations as not only an Assessment Tool but also a Learning Tool, In *Proceedings of the ACM SIGCSE Symposium*, 2006.

[10] Rodger, S., JFLAP, www.jflap.org, last accessed May 25, 2017.

[11] Sipser, M., *Introduction to the Theory of Computation*, third edition, Cengage Learning, 2012.

[12] VanDeGrift, T., Art in Theory of Computation, *Journal of Computing Sciences in Colleges*, 32 (1), 2016.

[13] Vygotsky, L.S., *Mind in Society: The development of higher psychological processes*, Harvard University Press, 1978.

[14] Walker, H.M., Some Strategies When Teaching Theory Courses, *ACM Inroads*, 5 (3), September, 2014.