Understanding and **P**ropagating the Essence of Successful **CS Ed**ucation Research Projects

Proceedings of the UP CS Ed Workshop at the 12th International Computing Education Research conference

Melbourne, Australia, September 8 – 12, 2016

Workshop Co-Organizers: Eileen Kraemer and Murali Sitaraman

Proceedings Editor: Aubrey Lawson

UP CS Ed Research

Understanding and **P**ropagating the Essence of Successful **CS Ed**ucation Research Projects

A 2013 report from a joint committee of the U.S. Department of Education(ED) and the US. National Science Foundation (NSF), Common Guidelines for Education Research and Development (NSF 13-126) describes shared understandings of the roles of various types of research in generating evidence about strategies and interventions for increasing student learning. They identify six types of research:

1. Foundational Research
2. Early-Stage or Exploratory Research
3. Design and Development Research
4. Efficacy Research
5. Effectiveness Research
6. Scale-up Research

For each type of research, the guidelines characterize the elements of such research: purpose, required theoretical and empirical justifications, expectations for research design and expected products, and expectations for review.

In this NSF-funded workshop we seek to explore the space of successful CS Education Research projects conducted by ICER attendees, identify exemplars in each of the six categories, and discuss how best to disseminate these guidelines to current and prospective researchers in the field. Outcomes of the workshop will include:

- e-proceedings
- Publications including in an ACM magazine and an ACM SIGCSE special session
- Topics for a workshop at SIGCSE for current and prospective CS Education researchers
- Other dissemination efforts

Workshop attendees will be asked to present and describe their past or current research project in terms of the elements in the guidelines (purpose, justifications, design, products, etc.) and to participate in discussions on how to promote the guidelines to prospective researchers.

UP CS Ed Research Workshop Organization

This workshop is organized by the Clemson Consortium for Computing & Education with support from the US National Science Foundation.

Organizing committee

Eileen Kraemer and Murali Sitaraman (co-chairs)
School of Computing, Clemson University
etkraem@clemson.edu, msitara@clemson.edu

Russell A. Marion II, S. Megan Che, Michelle Patrick Cook
Eugene T. Moore School of Education, Clemson University

Contents

Research Type: Effectiveness Study

Research Title: *Programming Problem Solving Pedagogy*

Authors: *Loksa, D., Ko, A.J., Jernigan, W., Oleson, A., Mendez, C., Burnett, M.M.*

Contact Information: *ajko@uw.edu*

Purpose

The goal of this study was to investigate the effectiveness of a new form of programming problem solving pedagogy on novice high school students' problem solving productivity.

Justification

Programming is a difficult task that it requires people to plan, organize, and reflect on their work in order to select appropriate strategies to make progress. Unfortunately, in most computing education settings, instructors provide little guidance or scaffolding on how to approach problem solving, leaving students to develop their own process. This causes many students to struggle to even solve simple problems. Prior work has investigated some approaches to teaching problem solving, including case studies and worked examples, but neither of these help students the type of self-awareness necessary to regulate their problem solving activities. In this project, we explored an approach that explicitly teaches learners about six distinct phases of programming problem solving, and develops learner awareness about which phase they are in, whether their current strategy is effective, and why. This approach sought to develop *self-regulation* skills in learners, improving their ability to select appropriate strategies for solving programming problems throughout their process.

Research Plan

To evaluate this approach, we offered two 2-week summer camps for high school aged learners who did not have prior programming experience. Forty-eight students enrolled in either a 3-hour morning or afternoon camp. In both camps, students learned the basics of HTML, CSS, and JavaScript, and were then given a set of requirements to meet for a personal website that was architected, but not implemented. Students were tasked with meeting these requirements over four days. The treatment group received two additional forms of instruction: 1) a 45-minute lecture on the six problem solving stages in programming, and 2) when students asked for help the TA would ask the student to explain what stage they were in, what strategy they were using, and whether it was working, and then provide the help provided to both groups. During the camp, we gathered daily measures of self-efficacy, growth mindset, help requests, and metacognitive awareness. At the end of the camp, we counted the number of requirements each student had satisfied each day and by the end of the camp.

Findings

The problem solving instruction and prompts significantly increased the number of requirements met, shifted help requests from design to debugging, led to significantly higher programming self-efficacy, and significantly higher growth mindset. Additionally, students in the experimental condition demonstrated significantly greater ability to recall and describe their problem solving process at the end of each day. This suggests that the mechanism by which problem solving productivity increased was via the predicted route of improved self-regulation. These significant differences emerged over the course of five-days of programming, suggesting that the intervention took time produce meaningful changes in each students' problem solving process. These results have direct implications for problem solving instruction in introductory programming courses.

Publications and Links to Relevant Webpages

Loksa, D., Ko, A. J., Jernigan, W., Oleson, A., Mendez, C. J., & Burnett, M. M. (2016). Programming, problem solving, and self-awareness: effects of explicit guidance. *ACM Conference on Human Factors in Computing Systems*, 1449-1461.

Research Type: Early-Stage or Exploratory Research

Research Title: Plagiarism and Related Issues in Assessments Not Involving Text

Author: Simon

Project co-authors: Beth Cook, Judy Sheard, Chris Johnson, Angela Carbone, Mario Minichiello, Chris Lawrence

Contact Information: simon@newcastle.edu.au

Purpose

The project set out to investigate academics' and students' understandings of academic integrity in assessment items not involving text; for example, computer programs and visual designs, as opposed to essays. The findings would be expected to inform the development of definitions and guidelines that would in turn inform the revision of policies and practices in academic integrity, in the recognition that what works for essays does not necessarily work equally well for other types of assessment. The ultimate goal would be educational approaches that would positively alter the academic practices of academics and students in computing and in other areas of study that use assessment items unlike essays.

Justification

The project was sparked by the researchers' observations that the principles and practices of academic integrity might not apply uniformly across all forms of assessment: that policies and examples couched in terms of direct and indirect quotation, paraphrasing, in-text references, reference lists, and use of the words of others, might not apply to such assessment items as a mathematical proof, a computer program, or a visual design. Likewise, the text-matching methods used as a first step in determining possible breaches of academic integrity might not be applicable to these non-textual assessment items.

If these observations were substantiated, it was likely either that academic integrity requirements were being ignored because they were deemed inappropriate, or that they were being applied in a literal manner that was in fact inappropriate for the type of assessment item in question. Either of these would be educationally unacceptable. The project therefore set out to determine whether the observations could be substantiated.

Research Plan

The project set out to answer the following questions about assessment items not involving text:

- What do academics and students in these areas think constitutes a breach of academic integrity?
- What do academics and students in these areas think might not be a breach of academic integrity, even though in a text-based area it might be?
- What do academics do to inform students as to the expected standard of integrity?
- What do academics do to detect similarities that might suggest academic misconduct? (For example, do they use automated tools or simply their own experience and awareness?)
- How do academics deal with academic misconduct when it is discovered?
- Are there areas (such as perhaps computer programming) in which academics and/or students think that there is only one correct answer, so copying cannot be detected?
- Are there areas (such as perhaps visual images) in which academics and/or students think that every answer is unique, so copying is acceptable so long as one personalises the copy?
- To what extent do academics in these areas believe that university policies for academic integrity based on text are adequate for non-text-based assessments?

In addition to a thorough literature review, the research design involved focus groups and a major online survey. The focus groups, to be conducted among academics and among students at three participating universities, would tease out the issues and inform the design of the survey. There is no guideline for the number of focus groups required for this purpose, but it was expected that findings from the literature, the researchers' own experiences, and six or more focus groups would adequately inform the survey design. The survey would be conducted online, with invitations sent to all universities in Australia. As the project was an Australian one, conducted with Australian government funding, that appeared to be appropriate coverage.

The focus group transcripts would be analysed by standard qualitative analysis procedures, and the survey data by standard qualitative and quantitative procedures.

Findings

The project made many findings. A few of the more interesting ones are listed here.

- Students and academics in computing and visual design believe that their university's academic integrity policies do not usefully apply to assessments in their disciplines.

- While there are standard ways in an essay of referencing material from external sources, there are no such standard ways in computing or in visual design.
- There is no universal agreement that referencing of external material is necessary in these disciplines as it is for essays.
- Students and academics in computing have significantly different perceptions regarding certain practices in essay assessments and the parallel practices in computing assessments.
- Students and academics in visual design have significantly different perceptions regarding certain practices in essay assessments and the parallel practices in design assessments.
- There are significant differences between the numbers who consider certain practices to be plagiarism or collusion and the numbers who consider those practices to be unacceptable.
- It is understood that designs are necessarily based on other designs, and there is no universally accepted requirement in design to identify and reference external sources of inspiration.
- It is generally expected that computing practitioners will reuse existing algorithms and code where possible, and there is no widely accepted requirement to reference material so used.

The findings certainly support the initial premise that further work is required to define and guide academic integrity practices and policies in areas of study that use assessment items not involving essays and similar prose works. That design and development work is beyond the scope of the original funded project, but is now proceeding separately.

Publications and Links to Relevant Webpages

Simon, Beth Cook, Judy Sheard, Angela Carbone, Chris Johnson (2013). Academic integrity: differences between programming assessments and essays. *13th International Conference on Computing Education Research – Koli Calling 2013*, Koli, Finland, November 2013, 23-32.

Simon, Beth Cook, Mario Minichiello, Chris Lawrence (2014). Academic integrity: differences between design assessments and essays. *Design Research Society annual conference*, Umeå, Sweden, 1260-1273.

Simon, Beth Cook, Angela Carbone, Chris Johnson, Chris Lawrence, Mario Minichiello, Judy Sheard (2014). How well do academic integrity policies and procedures apply to non-text assessments? *Sixth International Integrity and Plagiarism Conference (6IIPC)*, Gateshead, UK.

Simon, Beth Cook, Judy Sheard, Angela Carbone, Chris Johnson (2014). Student perceptions of the acceptability of various code-writing practices. *19th ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '14)*, Uppsala, Sweden, 105-110.

Simon, Beth Cook, Judy Sheard, Angela Carbone, Chris Johnson (2014). Academic integrity perceptions regarding computing assessments and essays. *Tenth International Computing Education Research Conference (ICER 2014)*, Glasgow, Scotland, 107-114.

Simon, Judy Sheard (2015). Academic integrity and professional integrity in computing education. *20th ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '15)*, Vilnius, Lithuania, 237-241.

Simon (2015). Academic integrity: it's not what they would have us believe. Keynote presentation at Sixth Annual CITRENZ Conference, Queenstown, New Zealand.

Simon, Judy Sheard (2015). In their own words: students and academics write about academic integrity. *15th International Conference on Computing Education Research – Koli Calling 2015*, Koli, Finland, 97-106.

Simon, Judy Sheard (2016). Computing assessments and academic integrity. *18th Australasian Computing Education Conference (ACE 2016)*, Canberra, Australia, February 2016, article 3.

Simon (2016). Academic integrity in non-text based disciplines. *Handbook of Academic Integrity*, 763-782 (Springer, Singapore).

Simon, Beth Cook, Judy Sheard, Chris Johnson, Angela Carbone, Chris Lawrence, Mario Minichiello (2016). Plagiarism and related issues in assessments not involving text. OLT final report, 2016. http://www.olt.gov.au/project-plagiarism-and-related-issues-assessments-not-involving-text-2012

Research Type: Design and Development Research

Research Title: Data-driven Support for Novice Programmers

Authors: Thomas W. Price, Tiffany Barnes

Contact Information: twprice@ncsu.edu; tmbarnes@ncsu.edu

Purpose

The goal of this project is to create and evaluate algorithms for generating data-driven hints and feedback for novice programmers working on open-ended assignments. Our research will result in an augmented novice programming environment that provides automated, customized feedback to students on their current program when they are stuck and unable to proceed on their own. This should allow students to continue working, even in the absence of an instructor or TA. We will evaluate our success by measuring the impact of our hints on novice students' performance and learning in an introductory CS setting.

Justification

CS instruction is arguably more effective now than ever, with new curricula that engage students' creativity and interests and programming environments that make CS more accessible through block-based editors [6]. However, even when we engage students and lessen the burden of syntax, programming remains a difficult subject to learn. When students struggle or get stuck, the burden falls almost entirely on instructors, who are not always available.

Intelligent Tutoring Systems (ITS) are adaptive learning tools that attempt to fill this gap by playing the role of a tutor, guiding a student's learning. ITSs offer support during problem solving, often with adaptive hints and feedback. Like many tutor interventions, these hints can be seen as scaffolding to keep the student in the Zone of Proximal Development [3]. Empirically, this approach has been shown to improve student performance both inside the tutor and on subsequent assessments [5].

Our work attempts to make these ITS-style hints available in existing, effective novice programming environments and curricula. Our data-driven algorithms automate the process of hint generation, lowering the barrier to use. We aim to improve on the state of the art in hint generation by designing an algorithm specifically for the open-ended programming assignments used in many introductory courses, which pose difficulties for existing techniques [2].

Research Plan

Our methods for achieving our research goals are as follows:

1) Instrument a programming environment (Snap*!*) and collect data from novices in a class setting as they work on introductory assignments. [Complete]
2) Design and iteratively refine an algorithm which adapts existing techniques for data-driven hint generation to the context of open-ended programming assignments. [Complete]
3) Perform a technical evaluation of the algorithm to determine the feasibility of our approach using historical data to generate and test hints. [Complete]
4) Run a small pilot study in a CS classroom and collect qualitative data on how students interact with available hints on a single assignment. [Complete]
5) Revise the algorithm to address the problems observed in Step 4.
6) Run a larger pilot study over a semester in a CS class, and compare student outcomes to historical data collected in Step 1.

The population we are studying is an introductory CS course for non-majors, consisting of 60-80 students per semester. In Step 6, our measures will be students' performance on assignments with hints available, their performance on a subsequent assignment without hints available and their performance on in-class assessments.

Findings

Our research so far has yielded three primary findings:

Existing hint-generation techniques are not well suited for open-ended programming assignments [2]. The data collected during the fall and spring semesters suggests that there is very little exact overlap among student solution, making it very difficult to apply traditional data-driven techniques. Additionally, open-ended assignments often have objectives that involve user interaction or visual output, making them difficult to asses automatically, which many existing techniques rely upon.

Despite these challenges, we can reliably generate data-driven hints [1]. A technical evaluation of the CTD algorithm shows that even on an open-ended assignment with almost no direct student overlap, the algorithm is still able to reliably generate hints which lead a student to a complete or nearly-complete solution.

These hints have potential to provide great benefit, but there are many challenges to successful implementation. Qualitative analysis of the pilot study reveals that the hints perform ideally for some students, but many use them rarely or not at all. Some of the generated hints were technically correct but still confusing to students. Other hints were quite reasonable but still ignored, indicating the need for further refinement of the algorithm and further study of what makes a good hint.

Publications and Links to Relevant Webpages

Demo page: http://go.ncsu.edu/isnap

Publications:

[1] T. W. Price, Y. Dong, and T. Barnes, "Generating Data-driven Hints for Open-ended Programming," in *Proceedings of the International Conference on Educational Data Mining*, 2016.

[2] T. W. Price and T. Barnes, "An Exploration of Data-Driven Hint Generation in an Open-Ended Programming Problem," in *Proceedings of the Workshop on Graph-Based Data Mining held at EDM'15*, 2015.

Other References:

[3] T. Murray and I. Arroyo, "Toward Measuring and Maintaining the Zone of Proximal Development in Adaptive Instructional Systems," in *International Conference on Intelligent Tutoring Systems*, 2002.

[4] D. Garcia, B. Harvey, and T. Barnes, "The Beauty and Joy of Computing," *ACM Inroads*, vol. 6, no. 4, pp. 71–79, 2015.

[5] A. Corbett and J. Anderson, "Locus of Feedback Control in Computer-Based Tutoring: Impact on Learning Rate, Achievement and Attitudes," in *Proceedings of the SIGCHI Conference on Human Computer Interaction*, 2001, pp. 245–252.

[6] T. W. Price and T. Barnes, "An Exploration of Data-Driven Hint Generation in an Open-Ended Programming Problem," in *Proceedings of the Workshop on Graph-Based Data Mining held at EDM'15*, 2015.

Research Type: Early-Stage or Exploratory Research

Research Title: *Automated Tutor for Pinpointing Code Reasoning Obstacles and Improving Student Understanding*

Authors: *Michelle Cook[1], Jason O. Hallstrom[2], Joseph E. Hollingsworth[3], Matthew Pfiste[4], and Murali Sitaraman[4]*

1 College of Education, Clemson University, Clemson, SC 29634

2 Computer and Electrical Engineering, and Computer Science, Florida Atlantic University, Boca Raton, FL 33431

3 Computer Science, Indiana University Southeast, New Albany, IN 47150

4 School of Computing, Clemson University, Clemson, SC 29634

Contact Information: *murali@clemson.edu*

Purpose

The project goals are to pinpoint fine-grain obstacles that students face in reasoning correctly about code compositions and to help improve their code understanding through tailored activities. The approach involves the use of a logical reasoning tutor, aided by an automated verification engine that checks code correctness. The verification engine makes it possible for the tutor to offer a class of learner activities and directed logical feedback not possible with standard development environments, such as Eclipse. The project aims to address both beginning-level programming constructs and compositions, such as assignment and if-then-else statements, and more advanced software engineering concepts, such as component contracts and invariants.

Justification

The significance of the project is that it will enhance the logical reasoning ability of students, and thus improve the quality of the software they develop and maintain after graduation. We have previously shown that undergraduate students can learn to reason formally about code correctness [Drachova 2015]. Traditional reasoning methods include having students run their programs on select inputs, and then study the outputs to determine program behavior, both wanted and unwanted. This approach often gives students a limited understanding of a program's behavior—i.e., only what it does on selected inputs. We seek to enhance this approach by helping students reason analytically about program behavior over *all* inputs, including untested inputs. Furthermore,

traditional teaching methods require enormous assessment effort on the part of instructors to pinpoint a student's reasoning obstacles and provide specific feedback. Through automation, the logical reasoning tutor aims to overcome these challenges and improve code reasoning for individuals and groups. A precursor to the logical reasoning tutor is a web environment for developing and reasoning about code [Cook 2013].

Research Plan

While the research plan for the project considers questions of engagement and impact on different subsets of populations, in this description we focus on the following *research questions*: Does the logical reasoning tutor help pinpoint individual and collective fine-grain reasoning obstacles, and can it enhance student understanding? By fine-grain, we mean going beyond traditional identification of broader themes, such as difficulties with understanding assertions, method calls, or loops, and detecting if students understand composition of self-referential assignments and changes in values of variables, for example. The *research design* involves hypothesizing potential reasoning obstacles and devising activities that help to make fine-grain obstacles explicit to instructors. When a student-supplied answer is incorrect, the logical reasoning tutor is automatically capable of pinning down the reason for the incorrect answer by introducing subsequent student activities, each of which explores subordinate reasoning obstacles. The logical reasoning tutor will collect data on individual and group obstacles to facilitate individual and collective help.

The research will be conducted in the *context* of introductory and software development courses, where student reasoning obstacles will span beginning-level programming constructs and compositions, such as assignments and if-then-else statements, and more advanced software engineering concepts, such as objects; the latter includes both interface contracts and invariants. The collected data might inform us of fundamental fine-grain obstacles that have been difficult to pin down in common compositions and potential new obstacles in the more advanced object-based reasoning context. The tool will be instrumented to achieve fine-grain data collection on student inputs and verification outcomes.

To *validate* the thesis that the logical reasoning tutor, supported by a verification engine, can pinpoint fine-grain learning obstacles, we will gather data and analyze. For example, fine-grain data collected by the tool from a contract programming exercise might include the fact that an operation's implementation failed to satisfy the precondition of a called operation. In this example, the obstacle faced by the student might be one of several: the student fails to understand the basis for contract programming, fails to understand details of the mathematics used in the formal specification, fails to understand details about the calling operation's contracts, or something else. Unlike a classroom exercise where it might be hard to tease out a specific obstacle to understanding, it would be possible to pinpoint the obstacle with the logical reasoning tutor when it automatically provides subsequent exercises based on these other aspects of understanding. To ensure the *reliability* of the data, the tool will be employed in multiple courses at our institution as well as four others that have consented to experiment.

The logical reasoning tutor will be automated and will support instructor-extensible exercises to train learners to overcome their obstacles to understanding. The intervention provided by the tool is assistance to a learner that is tailored specifically to the obstacles detected by the tool on previous exercises. Improvements in reasoning with and without the help of the logical reasoning tutor will be compared and analyzed. *Data analysis* will be integrated into the tool. Reporting will include both the ability of the tool to help pinpoint obstacles for different kinds of reasoning activities, and its ability to offer effective feedback and improve student understanding for individuals and groups.

Findings

The project has just begun and there are no findings at this time.

Acknowledgments

This research is funded in part by US National Science Foundation grant EHR-1611714.

Publications and Links to Relevant Webpages

Charles T. Cook, Heather K. Harton, Hampton Smith, and Murali Sitaraman, "Specification Engineering and Modular Verification Using a Web-Integrated Verifying Compiler," Proc. 34th International Conference on Software Engineering, IEEE/ACM, 2012, 1379-1382.

Svetlana V. Drachova, Jason O. Hallstrom, Joseph E. Hollingsworth, Joan Krone, Rich Pak, and Murali Sitaraman. 2015. Teaching Mathematical Reasoning Principles for Software Correctness and Its Assessment. Trans. Comput. Educ. 15, 3, Article 15 (August 2015), 22 pages. DOI=10.1145/2716316 http://doi.acm.org/10.1145/2716316

RESOLVE Software Research and Education Site: www.cs.clemson.edu/group/resolve/

Research Type: Effectiveness Study

Research Title: ***IUSE: Design, Development, and Implementation Projects: Computational Creativity To Improve CS Education for CS and non-CS Undergraduates***

Authors: *Duane F. Shell, PhD, Leen-Kiat Soh PhD, Elizabeth Ingraham, PhD, Brian Moore, PhD, Stephen Ramsey, PhD*

Contact Information: *dshell2@unl.edu; lksoh@cse.unl.edu*

Purpose

Our long-term vision is to address the growing need for computationally savvy, creative thinkers and problem solvers by incorporating computational thinking and creative thinking into the undergraduate Computer Science and STEM curriculum. This vision includes STEM fields but also extends more broadly to the social sciences, arts, music, humanities, and vocational/technical training. The purpose of our IUSE project is to build on the innovation from a previous TUES Grant (DUE-1122956) "Integrating Computational and Creative Thinking (*IC2Think*)" to expand the dissemination and implementation of the Computational Creativity Exercises and gain more complete, nuanced understanding of the factors influencing implementation efficacy.

Justification

The "Rebuilding the Mosaic" (National Science Foundation 2011) report notes that addressing emerging issues in all fields will require utilization and management of large-scale databases, creativity in devising data-centric solutions to problems, and application of computational and computer tools and interdisciplinary efforts. Computational thinking and creativity are critical to addressing important societal problems and central to 21st century skills (National Research Council 2012). Our computational creativity exercises do not involve programming code, but instead provide opportunities to creatively solve problems and do tasks seemingly unrelated to CS but requiring a CS principle for solution. They are designed to foster development of creative competencies [2,3,6], by engaging multiple senses, requiring imaginative thought, presenting challenging problems and combining both individual and group efforts.

In previous studies during *IC2Think*, we identified increased learning related to completing incrementally more CCEs [2,3] and found increased learning in a quasi-experimental trial for an introductory CS class for engineering majors [6]. These findings form the foundation for the extensions of CCE deployment and research proposed in this project. We want to find out whether similar effects will remain when the exercises are scaled including implementation in a broader range of lower- and upper-division CS and non-CS courses with CCEs incorporated as regular assignments within the course to provide a test in as close as possible to normal classroom conditions.

Research Plan

The overall *study aim* is to produce a final suite of validated, dissemination ready CCEs. We are using design-based research methodology [9] integrating classroom intervention and testing in an iterative cycle with feedback of results to improve and refine the CCEs. *Research objectives* are to determine if CCE have similar effectiveness in lower- and upper-division CS courses and in CS and non-CS courses, how different student characteristics (demographics, motivation, self-regulation, ability) may impact CCE completion and effectiveness, and why CCEs are producing their effects on learning. Table 1 shows the measures used. In terms of the *research design methods and procedures*, motivation and strategic self-regulation/engagement measures, creative competencies, self-efficacy, and the knowledge test are collected on a web survey. Goal Orientation, FTPS connectedness, Perceived Instrumentality, Mindsets, and self-efficacy are assessed pre, mid, and post course. Affect and strategic self-regulated engagement are assessed mid and post course. Creative competencies are assessed pre and post course. The knowledge test is collected post-course. Table 2 shows some statistics on the data collected. The basic research design for testing CCE effectiveness is to compare student outcomes from classes during the implementation semester to the same class(es) in the control semester. This design was augmented with Propensity Score Matching to pair a student sample completing CCEs with a matched control semester group using motivation, self-regulation, and GPA. Sub-analyses compare lower division (freshmen and sophomore) courses and upper division (junior and senior) courses; beginning (freshmen and sophomore) or advanced students (junior and senior), CS and non-CS majors. Additional analyses examine associations of students' motivation and strategic self-regulated engagement with outcomes and CCE completion, including Profile Analysis.

Categories	Measures
Outcome Indicators	Students' course grade, CCE grades, and a computational thinking knowledge test, based on CS concepts and application of such concepts used in IC2Think
Transfer Outcome Indicator	Students' self-efficacy for using their CS knowledge creatively in their field
General Ability Indicator	Students' cumulative GPA
General Creative Competencies	*Epstein Creative Competencies Inventory* (ECCI) [1] and a course-specific adaptation of the ECCI developed by the researchers.
Motivation and Strategic Self-Regulation	A battery of survey instruments validated in prior studies [4,7,8]; Motivation assessments include (1) Goal Orientation, (2) Future Time Career Connectedness, (3) Perceived Instrumentality of the course, (4) Implicit Theories of Intelligence (Mindsets), (5) Career Aspirations in CS, and (6) affect or emotional reactions to the course. Students' strategic self-regulation and engagement are assessed with the Student Perceptions of Classroom Knowledge Building (SPOCK) measuring (1) metacognitive strategic self-regulation, (2) knowledge building/deep learning strategies, (3) lack of regulation, and (4) behavioral engagement (question asking, studying).

Table 1. Measures used in our studies.

Semester	Courses	Pre-Survey Sample	Post-Survey Sample
Fall 2014, Spring 2015	N = 17 (5 FR CS, 1 SO CS, 3 JR CS, 5 SR/Grad CS, 1 FR Art, 1 JR Art, 1 JR Music)	**Control Semester**: N = 975 (786 male, 187 female; 362 FR, 232 SO, 167 JR, 210 SR/Other)	N = 532 (411 male, 121 female) (55%)
Fall 2015	N = 7 (1 FR CS, 1 SO CS, 2 JR CS, 1 SR CE, 1 FR Art, 1 FR Info)	**Implementation Semester**: N = 376 (309 male, 64 female; 47 FR, 157 SO, 87 JR, 82 SR/Other)	N = 239 (191 male, 48 female) (64%)

Table 2. Statistics of the data collected. FR = Freshmen, SO = Sophomore, JR = Junior, SR = Senior.

Findings

Across two lower division (freshmen, sophomore) and three upper division (junior, senior) CS courses we *found a significant linear "dosage" effect on standardized course grades* ($F(4, 239) = 3.34$, $p = .011$, partial $Eta^2 = .053$; linear trend $p = .005$) *and knowledge test scores* ($F(4, 153) = 2.49$, $p = .045$, partial $Eta^2 = .061$; ; linear trend $p = .012$) *for completion of more CCEs*. The increases were *not* trivial. Students improved about a grade point for each addition CCE completed (C+ to B- to B to B+ to A from 0 to 4 completed CCEs). Students improved on the knowledge test by about one point for each additional CCE completed with those completing four CCEs scoring 50% higher than those not doing a CCE. There was a similar linear "dosage" effects in both lower division and upper division courses. In a second analysis examining those students who did two or more CCE's, we used Propensity Score Matching based on student GPA and the motivation and strategic self-regulated engagement measures collected in the pre and post surveys to identify a matched control group from the students in the same courses from the Fall 2014 and Spring 2015 control semesters. *Students completing the CCEs in the implementation semester had higher z-score standardized class grades* (Implement $M = .42$; Control $M = 0.01$, $t = -3.78$, $p < .01$, Cohen's $d = .77$), *higher knowledge test scores* (Implement $M = 7.84$; Control $M = 6.68$, $t = -2.34$, $p = .02$, Cohen's $d = .35$), *and higher self-efficacy* (Implement $M = 70.88$; Control $M = 63.11$, $t = -2.65$, $p = .01$, Cohen's $d = .39$). There was no difference, however, in creative competency scores ($t = .28$, $p = .78$, Cohen's $d = -.05$). These effects were the same in both lower- and upper-division courses. Other studies have found associations between initial motivation and change across the semester in students' perceived instrumentality, and career aspirations and grades and knowledge test scores [5, 8].

References, Publications, and Links to Relevant Webpages

[1] Epstein, R., Schmidt, S., Warfel, R. 2008. Measuring and Training Creativity Competencies: Validation of a New Test. *Creativity Research Journal, 20*:7-12.

*[2] Miller, L.D., Soh, L.-K., Chiriacescu, V., Ingraham, E., Shell, D. F., and Hazley, M. P. 2013. Improving Learning of Computational Thinking using Creative Thinking Exercises in CS-1 Computer Science Courses. In *Proc. FIE* (Oklahoma City, OK, October 23-26), pp. 1426-1432.

*[3] Miller, L.D., Soh, L.K., Chiriacescu, V., Ingraham, E., Shell, D.F., and Hazley, M.P. 2014. Integrating computational and creative thinking to improve learning and performance in CS1. In *Proc. SIGCSE* (Atlanta, GA, March 5-8), pp. 475-480.

*[4] Nelson, K.G., Shell, D.F., Husman, J., Fishman, E.J., and Soh, L.K. 2015. Motivational and self-regulated learning profiles of students taking a foundational engineering course. *J Eng Educ., 104*, 74-100. DOI= 10.1002/jee.20066

*[5] Peteranetz, M. S., Flanigan, A. E., Shell, D. F., and Soh, L.-K. (in press). Perceived instrumentality and career aspirations in CS1 courses: Change and relationships with achievement. In *Proc. ICER* (Melbourne, VC, Australia, September 9-11).

*[6] Shell, D. F., Hazley, M. P., Soh, L.-K., Miller, L. D., Chiriacescu, V. and Ingraham, E. 2014. Improving learning of computational thinking using computational creativity exercises in a college CS1 computer science course for engineers. In *Proc. FIE* (Madrid, Spain, October 22-25, 2014), pp. 3029-3036.

*[7] Shell, D.F. and Soh, L.K. 2013. Profiles of motivated self-regulation in college computer science courses: Differences in major versus required non-major courses. *J. of Sci. Ed. and Tech., 22* (Feb. 2013), 899-913. DOI= 10.1007/s10956-013-9437-9

*[8] Shell, D. F., Soh, L.-K., Flanigan, A. E., and Peteranetz, M. S. 2016. Students' initial course motivation and their achievement and retention in college CS1 courses. In *Proc.* SIGCSE (Memphis, TN, March 2-5), 639-644).

[9] Wang, F. and M. J. Hannafin (2005). Design-Based Research and Technology-Enhanced Learning Environments. *Educational Technology Research and Development, 53*(4):5-23.

**Project Publications*

Project Web Site

http://cse.unl.edu/agents/ic2think/

Research Type: Design and Development Research

Research Title: *Gidget – A game for computing education*

Authors: *Michael J. Lee, PhD*

Contact Information: *mjlee@njit.edu*

Purpose

Gidget[1] is a freely available online educational game designed to teach users introductory computer programming (CS1) concepts [5]. Players must help the eponymous protagonist of the game—a damaged robot—solve debugging puzzles (i.e., fix broken code) to complete its missions. Each mission (i.e., level) has a specific learning objective, with the entire collection of game levels comprising a simple CS1 curriculum [10]. We envision Gidget as a low-barrier, entertaining medium for novices to gain exposure and experience with computer programming. Moreover, our goal is to keep users engaged with the game [9, 8, 10], and for them to show measurable learning outcomes [7].

Justification

The web is dramatically changing how, where, and with whom people learn. The largest and most accessible repositories for learning are no longer schools and libraries, but websites such as Wikipedia, Udacity, and Coursera[2]. These new types of discretionary learning environments allow learners to work at their own pace [20], engage with the material, and socialize with others – all at a scale that is typically unavailable in more compulsory learning settings. However, these online systems are relatively new pedagogical tools, and more research needs to be done to learn how to use them most effectively.

Educational games are a medium within the online learning space with large potential. Gaming can be used to provide a low-pressure, non-threatening, and engaging medium to learn new skills such as programming [15]. Games are now widely thought of as effective instructional tools [13, 16, 18] because they can share the attributes of a good teacher: they provide immediate feedback of success of failures, assist in learning at different rates, and offer opportunities to practice [14]. In addition, games can attract a wide and diverse audience. While traditionally viewed as a hobby for young boys [11], games are now a universal form of play, with 42% of females of all ages playing games [12, 19]. The average gamer is 37 years old, approximately 95% of US children ages 2-17 play video games [12, 19, 17], and the number of 55+ year old gamers is continuing to rise [19, 17].

[1] www.helpgidget.org
[2] www.wikipedia.org, www.udacity.com, www.coursera.org

Unfortunately, little is known about how to effectively engage and teach novices a CS1 curriculum at scale using online educational games. Moreover, we do not know much about who are playing these games, how successful they are, or how often they return. With the importance of computing education in the 21st century, along with the the rising number of internet users and increasing popularity of online learning resources, it is essential to research, develop, and test new ways to engage and teach effectively at scale.

Research Plan

We developed Gidget to be accessible and appealing to a wide range of users; this included using a gender-inclusive design approach [1, 6]. In addition to user-provided demographic information, Gidget also automatically collects all user code edits, interface interactions, and time logs. To date, Gidget's users are between 6 years old (with help from an adult) and 75 years old, with 46% of the registered users being female [5]. Players are predominately from the USA and Russia, followed by the U.K. and Canada.

Gidget was developed using an iterative development process, with each major release designed to answer a specific research question. Many of these releases featured a controlled experimental design, where users will split into different groups to test hypotheses about user engagement and/or learning outcomes. Data was collected in the form of pre and post tests of knowledge, pre and post questionnaires, and automated logging. In addition to running controlled experiments with hundreds of online users, we ran studies in the form of multiple summer camps for teenagers.

Findings

We have published on several of our key findings throughout our iterative design process while developing Gidget. Our studies have found that learners stay engaged with the game when: 1) a personified computer character conveys error messages [9]; 2) game goals are purposeful [8]; and 3) there are in-game assessments [10]. In addition, our studies have shown that leaners show measurable learning gains after playing through the game [7]. Moreover, Gidget is appealing to a wide audience—including teenagers from urban and rural populations [4, 6]—and can improve adults' initially negative views towards computer programming [2].

Since its public release, Gidget has attracted thousands of people from all over the world. It has been used successfully in many summer camps for high school and college students (especially focused on increasing females' participation in STEM), and there are plans to implement the entire game into semester-long, college introductory programming courses. Educators interested in using the game for their own classroom, enrichment program, and/or personal use are encouraged to do

so, and to contact the author for additional support. Researchers interested in using Gidget's log data for analyses are encouraged to contact the author for more information.

Publications and Links to Relevant Webpages

1. *Burnett, M., Churchill, E., Lee, M.J. (2015). SIG: Gender-Inclusive Software: What We Know About Building It. ACM CHI Extended Abstracts, 857-860.*
2. *Charters, P., Lee, M.J., Ko, A.J., and Loksa, D. (2013). Challenging Stereotypes and Changing Attitudes: The Effect of a Brief Programming Encounter on Adults' Attitudes Toward Programming.*
   ACM SIGCSE, 653-658.
3. *Gidget: www.helpgidget.org*
4. *Jernigan, W., Horvath, A., Lee, M.J., Burnett, M., Cuilty, T., Kuttal, S.K., Peters, A., Kwan, I., Bahmani, F., and Ko, A.J. (2015). A Principled Evaluation for a Principled Idea Garden. IEEE VL/HCC, 235-243.*
5. *Lee, M.J. (2015). Teaching and Engaging with Debugging Puzzles. University of Washington Dissertation (UW), Seattle, WA.*
6. *Lee, M.J., Bahmani, F., Kwan, I., Laferte, J., Charters, P., Horvath, A., Luor, F., Cao, J., Law, C., Beswetherick, M., Long, S., Burnett, M., and Ko, A.J. (2014). Principles of a Debugging-First Puzzle Game for Computing Education. IEEE VL/HCC, 57-64.*
7. *Lee, M.J., and Ko, A.J. (2015). Comparing the Effectiveness of Online Learning Approaches on CS1 Learning Outcomes. ACM ICER, 237-246.*
8. *Lee, M.J., and Ko, A.J. (2012). Investigating the Role of Purposeful Goals on Novices' Engagement in a Programming Game. IEEE VL/HCC, 163-166.*
9. *Lee, M.J. and Ko, A.J. (2011). Personifying Programming Tool Feedback Improves Novice Programmers' Learning. ACM ICER, 109-116.*
10. *Lee, M.J., Ko, A.J., and Kwan, I. (2013). In-Game Assessments Increase Novice Programmers' Engagement and Level Completion Speed. ACM ICER, 153-160.*

References

11. *Cassell, J. & Jenkins, H., eds. (1998) From Barbie to Mortal Kombat: Gender and Computer Games. MIT Press, Cambridge, MA.*
12. *ESA (2011). Essential facts about the computer and video game industry. Entertainment Software Association. http://www.theesa.com/facts/pdfs/ESA_EF_2011.pdf, retrieved August 21st, 2016.*
13. *Gee, J.P. (2007). What video games have to teach us about learning and literacy. Macmillan.*
14. *Gentile, D.A. (2009). Video Games Affect the Brain—for Better and Worse. The DANA Foundation, Cerebrum. July 23, 2009.*
15. *Griffiths, M.D. (1997). Video games: the good news. Education and Health, 15:10–12.*
16. *Hämäläinen,R., Manninen,T., Järvela, S., & Häkkinen,P. (2006). Learning to collaborate: Designing collaboration in a 3-D game environment. Internet and Higher Education, 9(1), 47–61.*
17. *Ito, M., Baumer, S., Bittanti, M., boyd, d., Cody, R., Herr B., Horst, H.A., Lange, P.G., Mahendran, D., Martinez, K., Pascoe, C.J., Perkel, D., Robinson, L., Sims, C., and Tripp, L. (2009). Hanging Out, Messing Around, Geeking Out: Living and Learning with New Media. Cambridge: MIT Press.*
18. *McGonigal, J. (2011). Reality is broken: Why games make us better and how they change the world. Penguin.*
19. *newzoo.com (2011). "High-level Game Facts from the US National Gamers Survey," retrieved August 21st, 2016.*
20. *Steffe, L.P., & Gale, J. E. (Eds.). (1995). Constructivism in education. Hillsdale, NJ: Lawrence Erlbaum, 159.*

Research Type: Early-Stage or Exploratory Research

Research Title: Applying Complexity Leadership Theory to the Adoption of Active Learning Practices

Authors: Eileen Kraemer, Murali Sitaraman, Russ Marion, Cazembe Kennedy, Gemma Jiang

Contact Information: {etkraem,murali,marion2,cazembk,xiaoyaj}@clemson.edu

Purpose

Recent work by researchers in computing education has established the use of active learning techniques as a pedagogical best practice, with the dual benefits of engaging students across the board in learning and broadening participation in computing [Borrego 2014]. However, widespread adoption of active learning strategies by the faculty of an academic unit remains a stubborn challenge [Olson 2012]. The purpose of this research is to evaluate the use of complexity leadership theory techniques to structure the context of an academic unit so that increased use of active learning emerges.

Justification

Complexity leadership theory (CLT) is a theory of organizational change that has been successfully applied in business settings, and is recommended as a promising approach by researchers working in change strategies for STEM disciplines in higher education [Uhl-Bien 2008]. CLT is based on the documented premise that synergistic dynamics of group synchrony (processing information interdependently within groups) is a more powerful stimulus for change, creativity, and productivity than are more traditional approaches that involve the development of skills in individual agents. CLT strategies foster the development of informal leaders and information flow across a system, with support from a formal leader. In this approach, network models are constructed that represent the entities and interactions in the unit under study, and then analyzed to produce measures that capture properties of the underlying system. Interventions may then be applied, and their effects evaluated in terms of the impact on the network and associated measures. The application of CLT has the potential to promote the emergence of active learning strategies in the complex system of an academic unit.

Research Plan

The *research question* for this study is: What are the risks and benefits of employing CLT in a higher education context? Specifically, we are conducting a feasibility study of CLT within the School of Computing at Clemson University. We are preparing to collect data that captures collaboration and interaction among the members of the School, and their beliefs, attitudes, and usage concerning active learning. We will use this data to perform network analyses that produce measures of social, advice, trust, task, resource, beliefs, knowledge and location networks to identify informal leaders

and information flows. These analyses will serve as the basis for identifying interventions that will influence the information flows, with the goal of measuring and propagating the use of active learning approaches. While identification of the actual interventions will be impacted by the outcomes of network analyses, they may include dissemination of active-learning materials, creation and deployment of a cadre of graduate assistants trained in the use of active learning techniques to assist faculty with their courses, establishment of learning communities, changes in department bylaws surrounding promotion and tenure, recommendations from a panel of experts, and other interventions that appear in the literature.

A central question for the feasibility of CLT in this context is the extent to which faculty are receptive to the concepts and participate in relevant activities. To gauge this, we will use various methods, including attitude surveys, interviews, and classroom observation.

In the context of this work, we will pilot a measurement scale that evaluates the degree of active learning in classrooms. We intend to evaluate what kinds of active learning measurements are realistically possible before fully developing and validating the scale with the intent that it be applicable to computer science instruction and easily modified for other fields, such as engineering or education. The scale will be used in analytical studies of the effects of network dynamics and network measures on classroom productivity.

An external advisory board of experts in active learning and Computer Science education will provide feedback and guidance. Additional feedback will be generated in the form of peer review and publications and conference presentations that result from the project.

Findings/Impact

The broader impacts of the proposed project will be significant. While the project will benefit all students, practices such as active learning methods are particularly beneficial for STEM students from disadvantaged backgrounds, students from underrepresented groups, and female students in male-dominated fields. However, for the benefits to be reaped, best practices need to be adopted into the culture of an organization. This proposal will inform the basis for making such changes happen and measuring the impact of such changes. Ultimately, the project can enable diffusion of best research practices to reach all STEM disciplines.

References

Borrego, M. & Henderson, C. (2014). Increasing the use of evidence-based teaching in stem higher education: A comparison of eight change strategies. *Journal of Engineering Education,* 103(2):22.

Friedrich, K., Sellers, S.L. & Burstyn, J. (2007). Thawing the chilly climate: Inclusive teaching resources for science, technology, engineering, and math. *To Improve the Academy: Resources for Faculty, Instructional, and Organizational Developmen*t, 26:133-144.

Olson, S. & Riordan, D. G. (2012). Engage to excel: Producing one million additional college graduates with degrees in science, technology, engineering, and mathematics. *Report to the President. Executive Office of the President.*

Uhl-Bien, M., Marion, R., & McKelvey, B. (2008). COMPLEXITY LEADERSHIP THEORY. *Complexity Leadership*, *5*, 185.

Research Type: Efficacy Study

Research Title: Case Studies of Programming Problems

Marcia Linn, PI: Mike Clancy, co-PI (author of this document)

Contact Information: clancy@cs.berkeley.edu

Purpose

The purpose of this NSF-supported work (MDR-8470514, MDR-8954753) was to design and evaluate course material that we call *case studies*, with the particular goal of determining how working with case studies can improve students' programming design and development skills.

Each of our case studies includes (a) a statement of a programming problem; (b) the commentary, a narrative description of an expert's solution written so a student can understand the expert's approach; (c) the worked-out solution in the form of the expert's code; (d) study questions to guide students in analyzing the program; and (e) test questions to assess students' understanding of the program solution. The narrative emphasizes the decisions encountered by the programmer and the criteria used to choose among alternatives.

Justification

Berkeley's computer science curriculum includes several project courses that include large programming assignments. Many students floundered with such assignments. It seemed clear that they were missing design and development skills for building and testing big programs; this was not a surprise, since there was little if any instruction in this area. We hoped that case studies would replace unguided discovery as a mechanism for learning these skills.

The case method was first used at Harvard College in 1870 and has permeated curricula for business, law, and medicine across the country. In the context of programming, case studies seem to have definite advantages.

- They model efficient ways to organize programming knowledge.
- They help students construct techniques and strategies that reduce or postpone the complexity of program design and development.
- They guide students to apply program design skills to large, complex problems and to learn context-dependent design strategies.
- They encourage students to reflect on completed solutions, comparing them to one another.

- They stimulate students to recognize their own strengths and weaknesses (in order to guard against the latter).
- They make large programs accessible to students, and thus give students a better picture of the nature of “real programming”.
- They form the basis for assessment of a student’s ability to design, understand, analyze, modify, and debug code that is both educational and effective.
- They provide a vehicle for active rather than passive learning, and exploration in teams.

Research Plan

We began by writing and piloting two case studies to use in subsequent work. One addressed the problem of printing block letters. Another involved designing and comparing two versions of a program to print a calendar. Our next steps are drawn from [1] and [2] and are summarized below.

We recruited instructors from San Francisco Bay Area high schools. Subjects came from ten Pascal programming classes in eight schools. Three teachers taught introductory Pascal, three teachers taught Advanced Placement (AP) Pascal, and two instructed at both levels. The total sample consisted of 121 students.

Guided by the high school teachers, we chose three instructional settings.

- The condition closest to teacher preferences was the "student solution + expert code" condition where students created their own solution and then studied the Pascal code from the expert's solution.
- In the "student solution + expert commentary + expert code" condition, students first created their own solution to the problem, and then studied the expert commentary and the Pascal code written by the expert.
- The condition least consistent with teacher preferences was the "expert commentary + expert code" condition where students did not create their own solution to the problem but did use the commentary and code.

The study design accounted for the variety in programming classes by using a baseline condition where students studied one case study in the condition that was in the middle with regard to teacher preference. As a result we could control for such factors as student backgrounds, teaching style, access to technology, or classroom conditions.

All classes used the two case studies mentioned above. First, for the baseline, all classes completed the Block Letters case study including student solution, expert commentary, expert code, and study questions. Classes were then randomly assigned to one of the three alternative approaches endorsed by the teachers for the Calendar case study: (a) student solution + expert commentary + expert code, and study questions; (b) expert commentary + expert code and study questions; and (c) student solution + expert code, and study questions. Comparing conditions with the expert commentary (a and b above) to those without the expert commentary (c above) allowed us to assess the impact of the commentary on program design skills.

To ensure that the students had the prerequisite background, teachers administered the case studies as soon as they had covered all the Pascal constructs used in the program. Thus, classes encountered each case study after varying amounts of classroom instruction. All classes completed the case studies by early spring. Teachers were free to introduce and deliver each case study in a manner that was typical and appropriate to them. Students could work together or individually while studying the commentary, running the program, or answering the study questions. Case study tests counted towards class grades. Students worked on the test questions alone, during regular class periods.

The case study tests assessed integrated understanding of program design skills but did not in any way depend on the expert commentary. Students had access to the worked-out solution to the problem during the test. Students were asked to engage in such design activities as reformulating the calendar program to accommodate a 6-day week.

Findings

Our findings were summed up in [2]:

> We found that students who received the expert commentary learned significantly more about program design than did students who received the expert code without the commentary. Posthoc comparison of the treatments revealed that the classes in both the conditions with the expert commentary significantly outperformed the other classes on the Calendar test: Wilcoxon Test ($W = 6$; $n = 3, 7$) probability < .03; Analysis of Variance ($y = 7.63$; $df = 1,7$; $SEy = 2.0$; $t = 3.82$) probability < .05. (For details of the statistical analysis, see [1].)
>
> Thus, the students using the expert commentary developed better design skills than the students who did not use the expert commentary. Also, students who implemented their own solution to the problem and then studied the expert commentary learned no more about designing the solution to a computer program than those who had the expert commentary alone. The students who did not study the commentary, i.e., who implemented their own solutions and then examined the expert code, were the least successful in this study. Given that precollege programming students normally try to avoid reading, these results provide strong evidence for the advantage of expert commentary.
>
> These results indicate that examining expert code for a computer program teaches students less about program design than does the expert commentary plus the code. Students need more assistance than just expert code in order to learn design skills. Even though the students examined the expert code and answered study questions intended to get them to infer the design process, this was not as effective as reading the expert commentary.
>
> These results also indicate that writing a computer program is less helpful than having expert commentary for developing design skills, even when test questions require application of patterns that were used for writing the program. Neither group of students who implemented their own solutions to the Calendar problem had an advantage over those who did not. The performance of the students who wrote the computer program is consistent with the hypothesis that designing the solution to one computer program does not impart an understanding of the generality of that solution. In contrast, examining an expert solution,

where alternative design decisions are contrasted, does contribute to understanding how to apply a pattern learned in the design of one program to a related program.

Follow-On

Work in the late 1980's and early 1990's involved the design of two collections of Pascal-based case studies [6, 7].

In 1990, Berkeley's introductory programming course was modified to use the Lisp language and to focus on functional programming. This required a complete reorganization of the course material and online environment. NSF's Advanced Technology Division funded a large part of this work. It yielded several case studies, plus a system for working with case studies online. The latter provided organized access to the various parts of the narrative, several mechanisms for assessment, a facility for testing individual procedures, and a "modeler" that controls the execution of the Lisp interpreter in a user-friendly way. The system is described in [4] and [5].

In the meantime, the Advanced Placement (AP) CS Test Development Committee had begun to consider how part of the AP CS exam could be based on a case study. (Clancy chaired this committee from 1987 to 1992.) One problem they faced was that concepts such as design and testing could not be tested without providing a context—impractical in an online setting. Moreover, they were concerned with the equivalence of the AP CS exams with those in college courses; in particular, college data structures courses generally involved significantly longer assignments than their high school counterparts. After an ETS pilot study and the authorship of two case studies, the decision was made in 1995 to base part of each exam on a case study. This lasted until the AP CS courses were reorganized in 2014.

Publications

1. "Can Expert Explanations Help Students Develop Program Design Skills?", Marcia C. Linn and Michael J. Clancy, *International Journal of Man-Machine Studies*, volume 36, number 4, pages 511-551 (1992).
2. "The Case for Case Studies of Programming Problems", Marcia C. Linn and Michael J. Clancy, *Communications of the ACM*, volume 35, number 3, pages 121-132 (1992).
3. "Case Studies in the Classroom", Michael J. Clancy and Marcia C. Linn, *Proceedings of the 23rd ACM SIGCSE Technical Symposium on Computer Science Education*, Kansas City, MO, March, 1992.
4. "Knowledge Integration in Introductory Programming: CodeProbe and Interactive Case Studies", John E. Bell, Marcia C. Linn, and Michael J. Clancy, *Interactive Learning Environments*, volume 4, number 1, pages 75-95 (1994).
5. "Can Tracing Tools Contribute to Programming Efficiency? The LISP Evaluation Modeler", Lydia M. Mann, Marcia C. Linn, and Michael J. Clancy, *Interactive Learning Environments*, volume 4, number 1, pages 96-113 (1994).

6. *Designing Pascal Solutions: A Case Study Approach*, Michael J. Clancy and Marcia C. Linn, W.H. Freeman and Company, 1992.
7. *Designing Pascal Solutions: Case Studies with Data Structures*, Michael J. Clancy and Marcia C. Linn, W.H. Freeman and Company, 1996.
8. Instructor's manual to accompany *Designing Pascal Solutions*, Michael J. Clancy and Marcia C. Linn, W.H. Freeman and Company, 1993.
9. *The AP Computer Science Directory Manager Case Study*, Michael Clancy, College Entrance Examination Board, 1992.
10. *A Teacher's Manual for the Directory Manager Case Study*, College Entrance Examination Board (Advanced Placement Computer Science), 1993.
11. *The AP Computer Science Marine Biology Case Study*, Michael Clancy, Owen Astrachan, and Cary Matsuoka, College Entrance Examination Board, 2000.

Research Type: Effectiveness Study

Research Title: *Effectiveness of Analogies in CS Education*

Authors: *Paul Cao, Leo Porter, Dan Zingaro*

Contact Information: *yic242@eng.ucsd.edu , CSE Department at UC-San Diego*

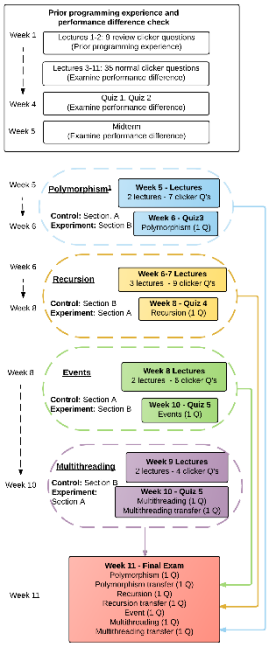
Purpose

The central research question of this study is: does introducing a topic with appropriate analogies improve student learning of that topic?

Justification

Analogies are considered a powerful instruction tool for improving student learning. We believe a significant percentage of instructors have used analogies when a new concept is introduced. Students usually appreciate this pedagogy when a connection is established between the concept covered in class and another entity they are familiar with. However, there is no solid evidence that analogies help student learning in computing. Existing research is scarce and prior results are mostly carried out in a laboratory environment.

Other CS education research related to analogies include proposing new analogies, analyzing existing analogies with a focus on their soundness, and analogy-related instructional design. Relevant experimental work in CS education research have focused on analogical encoding, a concept rooted in psychology. In analogical encoding, learners compare analogies across multiple domains, and try to capture the inherit relationships among common features from different domains.



The effectiveness of analogies in CS education are better measured in real classroom settings instead of laboratory environment. The time spent on analogies should not be extensive to mirror what happens in a normal class. Faculty member usually introduces a concept, then uses one or more analogies to make the ideas across before delving deeper into the content.

A better understanding on whether analogies work for students is beneficial to both instructors and students. Active research in this area will generate and crowdsource excellent analogies for every instructor to use. This in return will make classes more enjoyable to students.

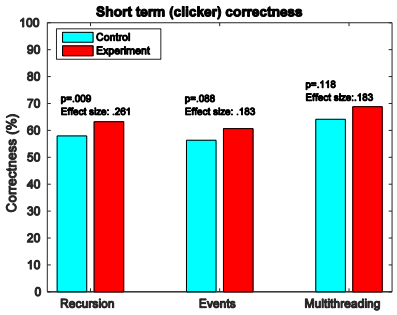
Research Plan

We estimated that if there is any effect on student learning from analogies, it will be fairly weak, and the benefit of analogies might vanish quickly. Thus we decided to have measurements based on two time scales: short term (days) and long term (weeks). The short term effect has to be measured in class and since we use peer instruction, we decided to use clicker questions. The long term effect will be measured in a quiz or exam. To ensure the best experimental conditions, there should be two classes that are taught by the same instructor in the same term. These two classes should be as similar as possible with respect to the time they are offered and student composition. However, it is impossible to control student composition and randomized assignment cannot work for this project. Therefore, we decide to alternate the two sections as experimental group and control group throughout this project. Thus the study isn't a strictly controlled experiment. We selected four topics and the experimental design is shown on the left.

To avoid potential confounding factors such as the background of students in the two sections, we compared the two sections' grade distributions before the start of the experiment and found no difference.

Findings

The conclusion of this study based on observed data is analogy has some meaningful benefits in the short term though the conclusion is not definitive. The following figure shows the comparison of control group and experiment group on three concepts.



No long term effect was detected in this study.

Publications and Links to Relevant Webpages

1. Yingjun Cao, Leo Porter, and Daniel Zingaro. Examining the Value of Analogies in Introductory Computing. In *Proceedings of the 2016 ACM Conference on International Computing Education Research*. 2016
2. J. D. Bransford, A. L. Brown, and R. R. Cocking. How people learn: Brain, mind, experience, and school. National Academy Press, 1999.

3. Y. S. Chee. Applying Gentner's theory of analogy to the teaching of computer programming. International Journal of Man-Machine Studies, 38(3):347-368, 1993.
4. D. Gentner. Structure-mapping: A theoretical framework for analogy. Cognitive Science, 7(2):155-170, 1983.
5. D. Gentner, J. Loewenstein, and L. Thompson. Learning and transfer: A general role for analogical encoding. Journal of Educational Psychology, 95(2):393-405, 2003.
6. S. Iyer and S. Murthy. Demystifying networking: Teaching non-majors via analogical problem-solving. Proceedings of the 44th ACM Technical Symposium on Computer Science Education, pages 77-82, 2013.
7. K. N. Macfarlane and B. T. Mynatt. A Study of an Advance Organizer As a Technique for Teaching Computer Programming Concepts. In Proceedings of the 19th ACM Technical Symposium on Computer Science Education, pages 240-243, 1988.
8. H. Neeman, L. Lee, J. Mullen, and G. Newman. Analogies for teaching parallel computing to inexperienced programmers. ACM SIGCSE Bulletin, 38(4):64, 2006.
9. J. P. Sanford, A. Tietz, S. Farooq, S. Guyer, and R. B. Shapiro. Metaphors we teach by. Proceedings of the 45th ACM technical symposium on Computer science education, pages 585-590, 2014.