Teaching Computational Thinking to Non-computing Majors Using Spreadsheet Functions

Kuo-Chuan (Martin) Yeh, Ying Xie, and Fengfeng Ke
martin.yeh@psu.edu, dr.ying.xie@gmail.com, fke@fsu.edu

Abstract - **Recently, higher education has seen an increasing emphasis on the prominent role of computational thinking in all disciplines. Computational thinking is advocated as not only a fundamental skill or concept in computer science but also a core competency for all disciplines. Teaching students in non-computer science majors computing thinking is challenging because students do not have experts' mental models. This study investigates the knowledge gap that non-computing major college students (n=126) possess about computational thinking in an introductory MS Excel course by measuring their performance using spreadsheet functions in three categories: recall, application, and problem solving. The empirical result, analyzed using ANOVA, shows that students can recall the meaning of those functions but seem to have trouble using them correctly and precisely (cued or uncued). Students' test results suggest the following issues: (1) problems with understanding the data type, (2) failure in translating problems to productive representations using spreadsheet functions, and (3) inadequate stipulation of the computational representations in precise forms. Addressing these problems early and explicitly in future classes could improve the education of computational thinking and alleviate difficulties students may experience in using computational thinking in learning and problem solving.**

Index Terms–Computational Thinking, Computer Science Education, Spreadsheet Functions, Problem Solving.

INTRODUCTION

Recently, computer scientists have stressed the prominent role of computational thinking (CT) in all disciplines and computer science educators should "make computational thinking the 21st century literacy" [1]. They argue that CT should be a core competency that provides students with a grounding for controlling and managing cognitive activities, which improves problem solving in all disciplines [2], [3]. The tasks people face at work every day are growing in scope and complexity. CT prepares learners in different disciplines for those challenges. As educators, we should include CT in our curriculum and facilitate CT learning at different times.

This paper begins with the definition of CT, describes the demographics of the study participants, our research design, and a description of the measurement instrument. It is followed by descriptive and statistical analyses in the results section. In the discussion section, the implication of the data and proposed future works are presented.

COMPUTATIONAL THINKING

CT may sound like a skill only useful in computer science. Yet, as Denning stated [3] "Computational thinking is part of computer science, but is not the whole story." Beyond computer programming, software engineering, or algorithms, CT is, more precisely, an ability or skill to analyze a problem, create abstraction, and solve it effectively [3]. This problem-solving process may or may not involve using technology. Knowing how to use a particular software application does not represent the entirety of CT. Advocates of CT note that CT consists of both mental tools and processes that differentiate CT from traditional computer programming skills.

One report generated by the Committee of the Workshops on Computational Thinking [3] has a great deal of discussions and descriptions about the nature and scope of CT. To illustrate the skill sets involved in CT, we now describe some characteristics of CT briefly.

- Automation of abstractions: Computational thinking focuses on the ability to manage complex situations by generating abstractions and maintaining the relationships among them.
- Precise representations: To generate abstractions, we need to have formal representations that reflect our cognitive processes and structures (discerning aspects of the situation).
- Systematic analysis: This characteristic of CT will enable us to generate hypothesis and search for a plausible solution systematically.
- Repetitive refinements: During problem solving, we consistently evaluate the current situation against our previous experience or our prediction until the best solution is reached.

I. Why Computational Thinking?

Modern technologies are so pervasive that computer literacy is no longer merely for a small number of computer and information technology professionals. Many jobs either require computing skills or benefit from CT in today's society. CT can help learners of all disciplines develop analytical skills and problem abstractions that help them solve their daily problems on the job. For instance, those

who use word processors should know that text content is separated from its format so that they can manipulate any portion without affecting others; those who use databases should know how data is processed behind the scenes so that they could better manipulate data without violating the implicit rules or create very expensive operations.

II. How to Teach Computational Thinking?

Computer scientists acquire CT through formal training at universities and experience on the job. CT of experts becomes tacit and spontaneous through everyday use. For example, experts naturally break a problem apart into smaller pieces and tackle them individually. Experts also readily repeat and refine a process until it is accurate and optimal. CT skills, however, can be difficult to acquire for novice learners, especially non-computing majors. This is oftentimes difficult because they do not possess or are not used to computational mindsets and analytical methods. Often non-computing majors' opportunistic (non-systematic) approach may not be the best way to manage and solve a complex problem.

Educators are increasingly incorporating CT curriculum into different grade levels. Lee and her colleagues reported their effort in promoting K-12 students in CT and proposed a three-stage progression (use->modify->create) for instructional designers [4]. Lu and Fletcher proposed to teach vocabularies and symbols (parts of the Computational Thinking Language) before CS students encounter their first programming course [5]. They suggested pre-college students would be better prepared for college if they are introduced to such symbolic representations.

To teach CT as a literacy skill to all disciplines, Guzdial [1] argues that computer science educators must understand why the novices struggle and where they struggle. To understand these problems and hence improve teaching and learning, we should collect formative data about learners and content. However, because computer scientists often take the fundamental knowledge of computing for granted, we often neglect concepts novices usually struggle with. In future teaching, mental tools and thinking processes should be taught explicitly to novice learners.

One essential CT skill is function abstraction [5] which computer scientists use almost every day directly and indirectly. The ability to use functions correctly and effectively requires several CT skills such as data representation, data process, abstraction, procedural thinking, etc. The use of functions by experts is so common that we have not paid enough attention to analyze and document how the concept of functions is internalized by novices or non-computing students, and what could be the problematic areas for learning functions among various aspects of CT. We thus look at functions in this study.

III. Our Research Design

Based on the characteristics of CT and the complexity levels of function learning, we parsed the learning of using spreadsheet functions into three categories: recall, application, and problem solving.

The recall category is rote memorization of function definitions and arguments. This is the lowest level of learning in a cognitive domain in Bloom's taxonomy of learning hierarchy [6]. In the application category, learners are presented with a set of data for which spreadsheet functions should be used to generate the right answers. In the problem solving category, learners are asked to solve a problem scenario using function(s) of their choice. The problem solving category, hence, is uncued where the learners have to search in their knowledge repository for a set of suitable functions to generate expected answer.

Each category requires different level of cognitive processes, which in turn can be facilitated by different instructional strategies. To teach the use of spreadsheet functions effectively, computer science educators need to understand the knowledge acquisition of all three categories and any underpinning problems when they experience trouble. This research study is to explore: (1) in which category novice learners struggle the most, and (2) the relationships among these categories.

METHOD

We recruited 126 non-computing major students who took a computer literacy course (introduction to spreadsheets and databases) that was required for many of their majors. The course was designed for second-year non-computing major college students, although our demographic data showed the majority were juniors. Participation in this study was voluntary and participants received 2% extra credit. Among all participants, two were in majors closely related to information technology (one was from Environmental Systems Engineering and the other one was from Management Information Science); other participants were from liberal arts, health and human development, or agricultural science. There were 13% underclassman (freshmen and sophomores) and 87% upperclassman.

A week after the spreadsheet section of the class was completed, participants used Microsoft Excel 2007 to answer questions of the three categories in an hour session. Questions of different categories were in separate spreadsheets, and they were instructed not to switch to the next spreadsheet until they completed all the questions on one sheet. The measurement was created based on the course content by the instructor and includes:

(1) Recall: participants were asked to explain the purpose/meaning of a function and its argument(s). They entered an open-ended description directly in a cell on the spreadsheet. Figure 1 shows an example recall question.

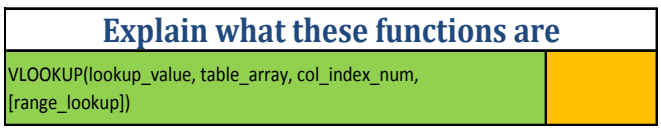

FIGURE 1
A RECALL QUESTION EXAMPLE

(2) Application: participants were asked to use data and a particular function to generate a correct answer. In this category, they were cued by what data was available and which function they should use. Figure 2 shows an example application question.

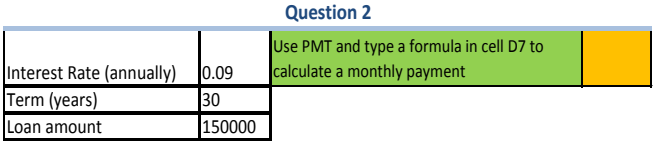

FIGURE 2
AN APPLICATION QUESTION EXAMPLE

(3) Problem solving: learners were asked to choose functions freely to solve two problems. There was no cue with regard to which function should be used. Figure 3 shows a problem solving example.

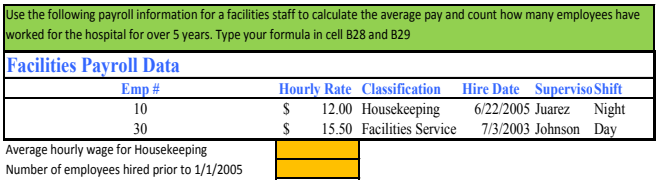

FIGURE 3
A PROBLEM SOLVING QUESTION EXAMPLE (ONLY TWO RECORDS ARE SHOWN IN HERE; THE ACTUAL QUESTION HAS MORE RECORDS)

Participants were instructed not to leave the Excel working environment—to prevent them from seeking help online. In addition, they were asked not to use Excel Help. They could, however, use the tooltip that showed function syntax when students enter function name by typing. They could also use the function argument dialog box for assistance.

The measurement instrument comprises: (a) 12 recall questions that count for a total of 24 points, with one point for the explanation of function and one point for the explanation of arguments; (b) 10 application questions (each counts for one point), and (c) two problem solving scenarios (nine functions required). The maximum scores for each category is 24, 10, and 9. This instrument was a workbook comprising multiple protected spreadsheets. Some of the cells in the spreadsheets were unlocked so that participants could only select and enter data, text, or formula into those unlocked cells.

The grading criterion for both application and problem-solving category is the following: a formula that contains a function must be used to generate the correct answer. Also students must use cell references rather than directly typing in data as function arguments.

RESULTS

Table I shows the basic statistics of students' performance in all three categories. Individuals' scores in each category were calculated by dividing the raw score by the maximum totals scores for its respective category. *All Functions* column includes all questions from our measurement instrument; *Non-stat Functions* column is generated using only the non-statistical functions in Excel 2007 from the measurement. The reason for excluding statistical functions is described in the following descriptions.

To answer the first research question—what category do non-computing students struggle with the most—a one-way ANOVA was used to compare the mean scores of all three categories. Our hypothesis was that when task complexity increased (problem-solving > application > recall), learners' performance would decrease. There was a significant difference on mean scores at the $p < .05$ level for these three categories ($F(2, 375) = 715.7$). A Tukey's HSD post-hoc analysis revealed that there was a significant difference between recall and application and between recall and problem solving. There was no significant difference between application and problem solving although the mean score of problem solving was higher than that of application. Because a basic mathematical course was a prerequisite for this course, the concept of summary, average, and count should be part of students' mental tools. Hence, students may already be familiar with basic statistical functions such as SUM, AVERAGE, and COUNT. To achieve a more accurate understanding of students' newly-learned knowledge during this course, we removed the questions of statistical functions (according to the categorization of Microsoft Excel 2007) when calculating students' scores, which generated another set of comparison (*Non-Stat Functions* column Table I). Based on the mean scores in the *Non-stat Functions* column, students' performance decreased consistently when task complexity increased. This finding was consistent with our hypothesis, suggesting that it was easier for students to recall the purpose and syntax of a function than to use the function with a given problem, without a hint or cue.

TABLE I
COMPARISON OF MEAN SCORES AND STANDARD DEVIATIONS (SD) AMONG THREE CATEGORIES.

Categories	All Functions M / SD*	Non-Stat Functions M / SD*
Recall	0.70 / 0.17	0.68 / 0.19
Application	0.41 / 0.19	0.36 / 0.20
Problem Solving	0.49 / 0.29	0.34 / 0.31

*M: Mean; SD: Standard Deviation

Table II shows paired t-test comparisons by categories. Students' performances in all three categories were significantly lower after common statistical functions were excluded from the calculation. It indicates that for a common-concept function (e.g. total, average, and count) that students are already familiar with, their chances of recalling its meaning, applying it to a problem, and using it in a problem solving situation are all higher than those of an unfamiliar function.

TABLE II
SIMPLE T-TEST BETWEEN ALL FUNCTIONS AND NON-STAT FUNCTIONS BY CATEGORY

Categories	t (df)	p
Recall	3.73 (249.96)	p <.005
Application	3.87 (249.52)	p <.005
Problem Solving	9.83 (202.82)	p <.005

To answer the second research question—how are these categories related to each other—three Pearson correlations on any two of three levels was calculated. Table III shows all three correlations are statistically significant: between recall and application, r = .38 (n=126, p<.05); between recall and problem-solving, r = .33 (n=126, p<.05); between application and problem solving r = .52 (n=126, p<.05).

TABLE III
PEARSON CORRELATION COEFFICIENTS (R) BETWEEN CATEGORIES AMONG RECALL, APPLICATION, AND PROBLEM SOLVING

Categories	Recall	Application
Problem Solving	.33	.52
Application	.38	

The Pearson correlation coefficients, shown in Table III, suggest that knowing the meaning of functions and their arguments helps students use functions and select an appropriate function for a problem solving scenario; the application score positively affect problem solving scores. That is, if students failed to apply a function correctly, it's very likely that they do not know which function to use or cannot use functions correctly in a problem solving scenario.

A further investigation of students' use of functions in both application category and problem solving category revealed that common errors included wrong type of arguments (numeric when text is required, or vice versa), missing arguments, wrong logic statements, and wrong function syntax. We also noticed that students either did not understand or forgot the purpose of double quotation marks—indicating any alphanumeric character embedded in between pairs of them should be treated as text data. Students' data representation in CT seemed weak. This is one of many reasons for which they could not use functions correctly. We also found that many participants did nothing more than the basic statistical functions in the problem solving category.

DISCUSSION

This study was to empirically investigate the challenges that non-computing major students faced in learning the use of spreadsheet functions. To a certain extent, our finding should also be applicable to novice learners when acquiring CT knowledge. Many researchers and educators in computer science community advocated that CT is an important skill for contemporary learners [1]-[3], [5], [7].

In our teaching experience, some areas of CT seemed especially difficult for non-computing majors. We created our measurement instrument based on two features of CT, application (abstraction of a problem) and problem solving, and added the recall (memory retention) category using spreadsheet functions to compare novices' performance in each category. We found that students' overall performance dropped significantly from recall to application and the performance in both application and problem solving categories were low—the correction rate was below 40% when statistical functions were excluded.

Our results show that even though students can recall the meaning of spreadsheet functions, they cannot use them correctly, with cue (application) and without cue (problem solving). Apparently, to formally represent a problem in computational format is still not within their mental skills, even after they are hinted to use a possibly productive function. When we further examined the source of their errors, their performance seemed to indicate that they have trouble differentiating different types of data or to follow the function syntax precisely. We hypothesized that when students are not familiar with the concept, the input data must follow an exact format; otherwise students will have difficulty recognizing them. In some disciplines, the outcome may be a continuum and sometime negotiable. In computer science, engineering, mathematics, and others alike, the science is to be precise. We also hypothesized that the students do not have an integrated mental representations about different type of data and how to deal with them. To such students, data, whether it is numeric or text, are simply combinations of alphabets that computers should be able to store and process; whereas numeric data are treated very differently by a computer than text data. Weak CT leads the students to believe that computers are like humans who can treat these data as the same.

Based on correlation coefficients of those three categories, recall scores positively correlate with both application and problem solving scores; application scores also positively correlate with problem solving scores. This result suggests that teaching should start with recall, but we need concentrate especially on application category, which seems to trouble most students. Better yet, the instructional strategies should encourage students to construct mental tools about computing. Pedagogical strategies such as problem-based learning [8] or case-based reasoning [9] may be useful for students to internalize CT skills and create individualized strategies fitting their styles.

In the problem-solving category, many students did not answer the non-statistical function parts. This fact demonstrates in new learning, transferring from rote memory to problem solving is still difficult for novice learners. In addition, using text data without double quotation marks suggests that novice learners rely heavily on what they already know to make sense of new knowledge. In most part of our daily experiences, there is no difference between "123" and 123. However, in a computational data representation, "123" is a text data that has no particular meaning and 123 is a numerical value that indicates a quantity. When teaching CT to non-computational majors, we must consider and perhaps pinpoint these preexisting mental representations and help learners overcome such problems early in their learning.

For future work, we should investigate the effect of a CT skill instruction before introducing spreadsheet functions on students' performances. In addition, computer science educators should examine whether CT skills help learning in other scientific domains including social sciences. Teaching CT in all disciplines, as is right now, is a concept well

accepted in the computer science education community. To make a broader impact on our society, this concept must be diffused to other disciplines so it can become a core of our education. This diffusion needs empirical data support.

A limitation of this work is the small number of questions in the problem solving category that had only five questions that used non-statistical functions. The measurement instrument can be improved by adding more functions from each category in the spreadsheet application in all three categories. This may help us analyze the exact computing thinking skills that are difficult to novice learners and why these skills are difficult for them. This will also help us understand the current status of CT in novices' knowledge and improve teaching CT more effectively.

CONCLUSION

Computational thinking is not about computer programming. Instead, it is a cognitive tool that helps us understand and solve problems. Nowadays, CT is an increasingly important skill for success in work and society where more and more tasks require problem solving skills. In addition, the ubiquity of technology also makes CT a vital competency like literacy and math. As a result, higher education has pronounced the emphasis to provide opportunities for students to enhance CT skills—CT is gaining much momentum rapidly.

Teaching CT to non-computing majors can be challenging. If CT were to be part of our core competency, we educators will have to conquer this challenge sooner or later. Understanding the learners and the tasks can be a pivotal step to making CT part of all curricula. This paper presents an empirical study that showed some of the weakness of novice learners. The findings provided us with the direction for our next step of teaching CT and showed a possibly effective approach to improve CT in non-computing majors.

ACKNOWLEDGMENTS

We would like to thank Yu-Hui Ching and Yu-Chang Hsu for helping us collect data. We also want to thank Robert St. Amant for providing useful information for the literature review. Frank Ritter gave comments that helped improve this paper.

REFERENCES

[1] M. Guzdial, "Education: Paving the way for computational thinking," *Communications of the ACM*, vol. 51, no. 8, pp. 25-27, 2008.

[2] J. M. Wing, "Computational thinking," *Communications of the ACM*, vol. 49, no. 3, pp. 33-35, 2006.

[3] Committee for the Workshops on Computational Thinking, National Research Council, *Report of a workshop on the scope and nature of computational thinking*. Washington, DC: National Academies Press, 2010.

[4] I. Lee et al., "Computational thinking for youth in practice," *ACM Inroads*, vol. 2, no. 1, pp. 32-37, 2011.

[5] J. J. Lu and G. H. L. Fletcher, "Thinking about computational thinking," *SIGCSE Bulletin*, vol. 41, no. 1, pp. 260-264, 2009.

[6] L. W. Anderson et al., Eds., *A taxonomy for learning, teaching, and assessing: a revision of Bloom's taxonomy of educational objectives*, Abridged. New York: Longman, 2001.

[7] S. Hambrusch, C. Hoffmann, J. T. Korb, M. Haugan, and A. L. Hosking, "A multidisciplinary approach towards computational thinking for science majors," *SIGCSE Bulletin*, vol. 41, no. 1, pp. 183-187, 2009.

[8] C. E. Hmelo-Silver, "Problem-based learning: What and how do students learn?," *Educational Psychology Review*, vol. 16, no. 3, pp. 235-266, 2004.

[9] J. L. Kolodner, *Case-Based reasoning*. San Mateo CA: Morgan Kaufman.

AUTHOR INFORMATION

Kuo-Chuan (Martin) Yeh, Assistant Professor, Department of Computer Science and Engineering, The Pennsylvania State University, martin.yeh@psu.edu

Ying Xie, Assistant Professor, Graduate Department of Educational Leadership and Instructional Design, Idaho State University, dr.ying.xie@gmail.com

Fengfeng Ke, Assistant Professor, Department of Educational Psychology and Learning Systems, The Florida State University, fke@fsu.edu