Cognitive Learning Strategies in an Introductory Computer Programming Course

Pruthikrai Mahatanankoon
pmahata@ilstu.edu

James Wolf
jrwolf@ilstu.edu

School of Information Technology
Illinois State University
Normal, IL 61761, USA

Abstract

Learning a computer programming language is typically one of the basic requirements of being an information technology (IT) major. While other studies previously investigate computer programming self-efficacy and grit, their relationships between "shallow" and "deep" learning (Miller et al., 1996) have not been thoroughly examined in the context of computer programming. Exploratory factor analyses using data collected from undergraduate information technology students, who just completed their first programming class shows distinct shallow and deep learning in computer programming. While shallow learning supports previous research, deep learning has three sub-scale activities: practice by examples, analytical thinking, and diagramming. The results also reveal that computer programming grit and self-efficacy have low to moderate correlations with shallow and deep learning, requiring further examination. Preliminary regression analyses also find that shallow learning positively influences computer programing grit and self-efficacy. Shallow learning strategies may be more widely employed during the initial stages of computer programming, while deep learning strategies may be more prevalent in higher-level computer programming courses. IT educators can examine this shift in strategies by observing students as they progress from introductory to advanced computer programming courses.

Keywords: Cognitive Learning, Programming, Deep Learning, Grit, Self-efficacy, Shallow Learning

1. INTRODUCTION

After over 50 years of study, the low success rates in introductory programming courses remain among the most intractable problems in computing education. The problem has been widely studied, but solutions have proven elusive. The lack of success in introductory programming courses and aversion to computer programing are often mentioned as significant factors in low retention numbers (McGettrick et al., 2005).

In addition to the large number of students with low performance, instructors often report a nearly equal number of students with high performance (Robins, 2010). As a result, instructors in introductory programming courses often report a binomial or two-humped distribution, with students grouped into the left and right tails, and few in the middle. This has led to a great deal of work that aims to understand the difference between these two student groups.

This reported binomial distribution has also led to a belief that programming is more innate than other academic subjects. That is that some people were born to be programmers while others are not. This belief is best captured in what Lister (2010) calls the idea of the "geek gene." According to this theory, those students born with the "geek gene" have the innate ability to program and the attitude necessary to succeed in

©2021 ISCAP (Information Systems and Computing Academic Professionals)

programming courses. Other students, presumably those born without the "geek gene", are genetically predestined to fail.

However, there is little empirical evidence to support this hypothesis. While most research finds that there is a correlation between mathematical aptitude and success in introductory programming courses, strong math skills are correlated with overall collegiate success (Pea and Kurland, 1984). Further programmer aptitude tests, like the PAT (Programmer Aptitude Test) administered by IBM and others, have shown little association with career programming success (Pea and Kurland, 1984). As a testament to its lack of predictive power, IBM no longer administers the PAT to prospective programmers (Lorenzen and Chang, 2006).

If innate skills and aptitude tests have little empirical evidence, then what could be the determinants of success in an introductory computer programming course? Recently researchers have begun examining cognitive factors and traits associated with student success in programming courses that are malleable and can be taught. For example, both grit and computer self-efficacy are associated with student success in programming courses (Wolf and Jia, 2015; Kanaparan, Cullen, & Mason, 2013; and Rex & Roth, 1998) are malleable (McClendon et al., 2017; Bandura, 1997). Similarly, student choice of cognitive learning strategies (deep or shallow) has been found to affect academic success and is changeable (Marton & Säljö, 1976b).

There has been limited research on the impact of grit, self-efficacy, and cognitive learning strategies on student success in introductory programming courses. This work will examine the relationship between these three constructs using scales developed explicitly for the task of computer programming. The study's purposes are 1) to test the measurement of cognitive learning strategies, i.e., shallow and deep learning, in the computer programming settings, and 2) to understand their relationships with grit and self-efficacy. Examining these relationships will help us extend the roles of grit and self-efficacy to students' cognitive learning strategies.

2. RELATED LITERATURE

Coding Grit

Grit is the trait-level perseverance and passion needed to obtain long-term goals (Duckworth et al., 2007). Grit is associated with academic success in a variety of academic settings (e.g., Duckworth and Quinn, 2009; Duckworth et al., 2007; & Strayhorn, 2013). Grit changes over time. People become gritter as they age, and their grit can be strengthened with deliberate practice (McClendon et al., 2017).

Grit has been widely studied in academic settings. For example, Duckworth et al. (2007) found that "grittier" students were more likely to succeed in both an Ivy League university and the United States Military Academy. Strayhorn (2013) found that African American males with higher grit earned higher grades in college than same-race male peers with lower grit. Similarly, Wolf and Jia (2015) found that grittier students earned higher grades in introductory programming courses than their less gritty peers.

While an abundance of studies demonstrates the positive link between intelligence and academic achievement (e.g., Laven, 1965), Duckworth et al. (2007) suggest that grit may be a better predictor of student success than talent or intelligence. Similarly, Wolf and Jia (2015) found that grit was a more powerful predictor of success in programming courses than college entrance exam scores.

While Wolf and Jia (2015) investigated the relationship between "generic grit" and student programming success, Mahatanankoon & Sikolia (2017) and Mahatanankoon (2018) altered the 12-point grit scale (Duckworth et al., 2007) to capture computer programming specific grit or coding grit. Mahatanankoon (2018) defined coding grit as the ability to "persevere and focus through computer programming activities." Mahatanankoon (2018) found that female computer science/information students were grittier. That is, they had higher levels of perseverance and long-term interest in computer programming than their male counterparts. In related work, Mahatanankoon & Sikolia (2017) found that passion and grit were positively correlated with computer programming attitude and retention in computer majors.

Computer Programming Self-efficacy

The widely studied information systems construct, computer self-efficacy, is an adaption of the more general self-efficacy (Compeau et al., 2006). Self-efficacy is four sources and reflects a future-oriented belief about one's ability to execute a specific task in a given context (Bandura, 1997). For example, computer self-efficacy is one's belief about their ability to use a computer (Compeau & Higgins, 1995). The four sources of self-efficacy beliefs are performance

accomplishment, vicarious experience, verbal persuasion, and physiological states (Bandura, 1977). As a result, self-efficacy is malleable.

Computer self-efficacy (CSE) is positively correlated with academic success in computer programming courses (e.g., Kanaparan, Cullen, & Mason, 2013 and Rex & Roth, 1998). Students with higher computer self-efficacy are more comfortable using computers and more confident in their computer-related skills. It is not surprising that this comfort and confidence leads to higher grades in computer programming courses.

In the seminal work in this area, Compeau & Higgins, 1995 defined computer self-efficacy (CSE) as the judgment of one's capabilities to use a computer in diverse situations. Mahatanankoon (2018) adapted the computer self-efficacy (CSE) (Compeau & Higgins, 1995) scale to capture self-efficacy for the specific task of computer programming. Computer programming self-efficacy or coding self-efficacy is one's belief about his/her computer programming ability. Mahatanankoon (2018) found that computer programing grit was a significant predictor of programming self-efficacy.

Deep and Shallow Learning Approaches

Marton, F., & Säljö, R. (1976a) identified two cognitive learning strategies: deep and surface. Within this framework, students adopt deep learning strategies when they intend to fully understand the subject matter and link it to their prior knowledge and personal experiences. In contrast, students adopt shallow learning strategies when their intention is merely to reproduce information without any further analysis (Murphy & Tyler, 2005).

Students using surface cognitive strategies are primarily concerned with storing the information into short-term memory, they focus on memory strategies (i.e., rote processing, repetition, reciting, and highlighting) (Boyle, Duffy, & Dunleavy, 2003). Students using surface cognitive strategies often rush through assignments and write down the first answer that comes to mind (Anderman, 1992).

Students using deep cognitive processing strategies try to understand new concepts by connecting new material with previously learned material, adopting a critical attitude towards information, and stopping to think about their work (Murphy & Tyler, 2005; Weinstein & Mayer, 1986). Students using deep cognitive processing strategies often monitor comprehension through self-quizzing, and engage in paraphrasing or summarizing (Anderman, 1992). In studies examining student achievement, several have found that academic performance was influenced positively by deep processing (e.g., Fenollar et al., 2007; Cano, 2005; Elliot et al., 1999; Miller et al., 1996).

As with earlier studies, we believe that the constructs under investigation, i.e., coding grit, coding self-efficacy, and cognitive processing, are related. The next section describes our methods.

3. METHODS

We conducted a field study to examine the relationships between coding grit, coding self-efficacy, and student cognitive learning strategies in introductory programming courses. The data were collected in fall 2019 (Sample 1) and spring 2020 (Sample 2). We used Sample 2 to verify Sample 1's results.

Sample 1

In fall 2019, we collected data from information technology students enrolled in systems analysis and design class, which had introductory Java programming as its prerequisite. Introductory Java programming is required for all IT majors in our department. We had 85 initial responses. After eliminating non-IT majors (n=2), telecommunication management (n=4), graduate MSIS students (n=13), duplications (n=6), and incomplete responses (n=7), we had a final total of 53 respondents in the study with Computer Science (32%), Information Systems (42%), and Cyber Security (26%). ANOVA also showed no significant mean differences in the research variables among the IT majors, i.e., Computer Science (n=17), Information Systems (n=22), and Cyber Security (n=14). Male students made up the majority of respondents (74%) in this sample. However, independent t-tests showed no significant mean difference in the research variables between male and female students.

Measures

For computer programming grit and computer programming self-efficacy, we will use previously developed scales (Mahatanankoon & Sikolia, 2017; Mahatanankoon, 2018). To examine shallow and deep learning, we modified the items developed by Greene and Miller (1993) to fit the context of computer programming. Appendix A shows the list of our modified questionnaire.

Analyses and Results of Sample 1

Exploratory factor analysis (EFA) was used to examine the dimensions of shallow and deep

learning in computer programming. Our initial factor analysis indicated five factors with eigenvalues higher than or equal to 1.0. Any factor with cross-loadings was eliminated. Through several iterations of EFA, Table 1 shows a three-factor solution with at least .50 loading value. The solution accounts for 62.7 percent of the total variance.

We see that shallow learning loaded into a single factor (Factor 1, SL1-SL4, DL11). We define *Factor 1* as the set of "fundamental" skills in computer programming, plus DL11 added to the factor. Deep learning skills loaded into two factors. *Factor 2* included DL3 and DL4 as the predominant items. DL3 involves working on several programming examples by repeating the same type of problems. Similarly, DL4 also relies on practicing and checking one's understanding of "new" concepts and rules. *Factor 3* entails analytical thinking of programming, which demands the ability to classify (DL6), analyze (DL7) coding problems, and eventually leading to an optimized solution.

Items	Factor 1	Factor 2	Factor 3
SL1	.636		
SL2	.971		
SL3	.699		
SL4	.540		
DL3		.618	
DL4		.988	
DL6			.917
DL7			.567
DL11	.590		

Table 1: Sample 1 Three-factor Solution

Sample 2

To verify our previous results, we collected another set of data using the students taking systems analysis and design, introduction to software and hardware concepts, and database systems in spring 2020. These courses have a Java programming class as a prerequisite. From our initial 62 responses, we eliminated three incomplete and four duplicated responses (n=7). Three graduate students, one telecommunication management, and three non-IT majors were dropped (n=7), giving us a total of 48 responses: Computer Science (n=11), Information Systems (n=17), and Cyber Security (n=20). This sample also has male students as the majority (87.5 percent). ANOVA found no significant mean differences in the research variables among the IT majors.

Analyses and Results of Sample 2

Our initial factor analysis resulted in four factors with eigenvalues higher than or equal to 1.0. Any factor with cross-loadings or with a loading value below .50 was eliminated. Through several iterations of EFA, Table 2 shows a three-factor solution with at least .50 loading value. The solution accounts for 60.6 percent of the total variance.

Our second sample had the same set of "fundamental" skills as our first sample. The results differed from Pilot Sample 1 in two aspects: 1) DL6 loaded onto Factor 2 as its third item, and 2) DL8 loaded with DL7 as Factor 3. These differences seem plausible, suggesting that the classification of programming problems (DL6) coincide with "practicing by examples" (Factor 2), and that finding a practical application could enhance "analytical thinking" (Factor 3).

Items	Factor 1	Factor 2	Factor 3
SL1	.622		
SL2	.870		
SL3	.655		
SL4	.544		
DL3		.867	
DL4		.799	
DL6		.652	
DL7			.665
DL8			.748

Table 2: Sample 2 Three-factor Solution

Earlier, we questioned the learning role of diagramming activities and included D1-4 during our data collection of Sample 2 (see Appendix A). Using the same EFA process with all previous items plus the new diagramming items (i.e., SL1-5, DL1-11, plus D1-4), the initial scree plot and eigenvalues revealed a five-factor solution. After several items were dropped due to low factor loadings (<.50) or cross-factor loadings, the final EFA iteration had a three-factor solution capturing 65.6 percent of the variability in learning. Table 3 shows the results of the factor loadings higher than .50.

Adding the diagramming items forced other deep learning strategies to load onto the same factor. However, without the diagramming items—dropping D3 and D4—the variability decreased to 58.9 percent (a 6.7 percent reduction) with a two-factor solution, as shown in Table 4.

The two-factor solution also persisted when we eliminated the diagram item (D1-D4) from the model (only Sample 2). To verify this result, we

force-loaded the same set of items using Sample 1, the loadings were similar to those of Sample 2, except for low factor loadings of DL2, DL6, and DL9.

Items	Factor 1	Factor 2	Factor 3
SL1	.634		
SL2	.903		
SL3	.792		
SL4	.624		
DL2		.566	
DL3		.766	
DL4		.718	
DL6		.757	
DL9		.750	
DL10		.732	
D1		.590	
D3			.936
D4			.785

Table 3: Sample 2 Three-factors Solution with Diagramming Items

Items	Sample 2 (n=48)		Sample 1 (n=53)	
	SL α=.826	DL α=.877	SL α=.832	DL α=.782
SL1	.641		.696	
SL2	.841		.903	
SL3	.880		.766	
SL4	.557		.548	
DL2^		.624^		--
DL3		.706		.639
DL4		.717		.988
DL6^		.801^		--
DL9^		.779^		--
DL10		.757		.523
D1^		.680^		NA
^ = excluded from composite reliability (α) calculation				

Table 4: Comparing Two-factors Solution of Both Samples

From our EFAs and results, shallow learning in computer programming meant that students focus on memorizing the solution or syntax without the tacit understanding of the logical sequences, concepts, and ideas behind coding. We can provide a list of activities constituting of what we called the "Shallow Learning" (SL-CP) in Computer Programming ($\alpha_{sample1}$=.832, $\alpha_{sample2}$=.826) as:

- I try to memorize the steps for solving programming problems presented in the text or in the lecture (SL1).
- When I study for the tests, I review my class notes and look at solved programming problems (SL2).
- When I study for tests, I used solved programming problems in my notes or in the book to help me memorize the 'programming' steps involved (SL3).
- I find reviewing previously solved programming problems to be a good way to study for a test (SL4).

On the contrary, "Deep Learning" in Computer Programming (DL-CP), from our factor analyses, constitutes *practice by examples*, although the learning activities varied between the two samples. Nevertheless, these recurring activities persisted among our respondents ($\alpha_{sample1}$=.782, $\alpha_{sample2}$=.877):

- I work on several programming examples of the same type of problems when studying this class so I can understand the problems better (DL3).
- I work practice programming problems to check my understanding of new concepts or rules (DL4).
- I work on practice programming questions/problems to check my understanding of new concepts or rules (DL10).

Besides, there could be *other* complementing activities for DL-CP. Based on the data, the sub-activities might include

a) DL-A Analytical Thinking ($\alpha_{sample1}$=.601, $\alpha_{sample2}$=.610)

- I classify programming problems into categories before I begin to work them (DL6).
- When I work a programming problem, I analyze it to see if there is more than one way to get the right solution (DL7).
- While learning new programming concepts, I try to think of practical applications (DL8).

b) DL-D Diagramming ($\alpha_{sample2}$=.816)

- I model different program modules or functions using some diagramming techniques (D3).
- I use some diagramming techniques to understand how programming work (D4).
- Some programming problems can be visualized using diagrams and models (D1).
- I draw pictures or diagrams to help me solve some programming problems (DL2).

Relationships with Coding Grit and Self-efficacy

On the one hand, self-efficacy is one's ability to control the outcome of a task-specific belief (Bandura, 1977). *Coding self-efficacy* (C-SE) is defined as "*one's belief about his/her computer programming ability" (Mahatanankoon, 2018, p. 2)*. It should enhance one's belief in the success of learning how to write computer programs. The higher the level of one's belief is, the more likely it is for the person to engage in computer programming. On the other hand, grit—a trait related to perseverance, passion, long-term commitment, and interest (Duckworth and Quinn, 2009)—may also be another internal factor driven by one's intention to enhance their knowledge and skills. Coding Grit (C-G) is defined as "one's ability to persevere and focus through computer programming activities" (Mahatanankoon, 2018, p. 2). Coding grit may encourage long-term learning interests in programming leading to both shallow and deep learning strategies. Therefore, to demonstrate nomological validity, we propose that coding self-efficacy (C-SE) and coding grit (C-G) that can be predicted by SL-CP, DL-CP, DL-Analytical Thinking (DL-A), and DL-Diagramming (DL-D, Sample 2 only).

Tables 5 and 6 reveal low correlations among our exploratory factors and the established measures. We see that coding grit and coding self-efficacy are moderately correlated. The cognitive learning strategies (i.e., SL-CP, DL-CP, DL-A, DL-D) are also moderately correlated, supporting the construct validity and suggesting that deep learning in computer programming are multidimensional (also see Tables 1-3).

	SL-CP	DL-CP	DL-A	C-G	C-SE
SL-CP	1				
DL-CP	.35	1			
DL-A	.28	.47	1		
C-G	-.13	-.18	-.12	1	
C-SE	.07	-.16	-.02	.59	1

Table 5: Sample 1 Correlation Matrix

	SL-CP	DL-CP	DL-A	DL-D	C-G	C-SE
SL-CP	1					
DL-CP	.44	1				
DL-A	.15	.48	1			
DL-D	.33	.59	.62	1		
C-G	.22	-.17	.01	.03	1	
C-SE	.18	-.30	-.11	.02	.50	1

Table 6: Sample 2 Correlation Matrix

From Table 7, we also explored the predictive validity (not hypothesized). The regression showed that the SL-CP positively predicted coding grit and coding self-efficacy. DL-CP, on the other hand, negatively predicted coding grit and coding self-efficacy. These significant findings were found only in Sample 2. In both samples, DL-A and DL-D did not influence the dependent variables.

4. DISCUSSION

Our study examined the measurement of shallow and deep learning in computer programming and tested the variable's relationships to coding grit and coding self-efficacy. EFA reveals the similarities of shallow learning in previous studies: route learning emphasized by memorizing and replicating the steps used to solve programming problems.

However, deep learning constitutes a multi-faceted construct. EFA solutions suggest at least three different activities: practicing, analyzing, and diagramming. Solving advanced programming problems calls for various viewpoints, which may be built on both shallow learning and higher cognitive strategies. Both samples yield inconsistent loadings. Future research warrants a larger sample size.

Our data leads us to question the importance of diagrams and models leading to programming solutions. From our factor analyses, the diagramming items (Sample 2) are not significantly loaded, although DL2 ("I draw pictures or diagrams to help me solve some programming problems') and D1 ("Some programming problems can be visualized using diagrams or models") correlated with a deeper level of learning (see Table 4). There are several plausible explanations:

1) The introductory programming class is the prerequisite of systems analysis and design, in which diagramming techniques are introduced. Therefore, diagramming is less valued by students taking programming for the first time.

2) Instructors have not emphasized a clear connection between the phases of analysis and design to the implementation (coding) activities.

3) Diagraming such as a flow chart or decision tree is used to conceptualize the program control statements, which is a *precursor* to introducing program syntaxes themselves. For example, the domain model class diagram assists the development of class definition (coding).

4) Our diagramming items (D1-4) are oversimplified and do not capture a wide variety

of diagramming activities. UML has a different set of modeling techniques that coincide with the different phases of the systems development life cycle.

The positive correlation between coding grit and coding self-efficacy is consistent with previous work. Similarly, shallow and deep learning strategies also have positive correlations among their designated items, but they are quite distinct from coding grit and coding self-efficacy, as our data have shown. Interestingly, deep learning in computer programming has an inverse relationship with coding grit and coding self-efficacy. We offer several explanations:

Firstly, shallow learning is an essential learning strategy to complete the class. Many students who completed an introductory programming course may focus on shallow learning to get a passing grade. Lizzio et al. (2002) find a positive link between a surface approach ("reproducing" approach with less knowledge integration) and a higher GPA among commerce students. Lizzio et al. (2002) posited that the given the narrower vocational focus of commerce courses and the typically employed assessment methods, surface methods, like shallow learning, may be a logical and strategic choice for students to pursue.

Secondly, deep learning strategies may occur in other advanced programming classes. Our samples are students who have just completed their first introductory programming class. Therefore, it is likely that deep learning has not been incorporated into their learning strategy. Computer programming skills may progress through different learning stages: Students use shallow learning to memorize language syntax, flow controls, and compilation steps. As they progress towards more advanced programming classes, evidence of deep learning strategies could be seen, including more substantial evidence of coding grit and coding self-efficacy. The research finding is mixed regarding the relationship between surface processing and academic performance, with most studies finding the relationship as either not statistically significant or negative (Watkins, 2001).

Thirdly, deep learning strategies may differ from one IT major to another, which affects the level of coding grit and coding self-efficacy required. Students usually begin with similar coding skills in the introductory programming course. As they progress to their intended information technology majors (e.g., computer science, information systems, cybersecurity, telecommunication, and others.), their programming needs and skills will adapt to their changing educational focus. Therefore, we may observe different types of deep learning, e.g., analytical thinking and diagramming, that differ across different IT majors. Echo this sentiment, Beattie et al. (1997) suggest that in certain academic situations adopting a surface approach may be advantageous. Fenollar et al. (2007) suggest that that memorization and rote rehearsal might be appropriate for some types of material and exam formats.

Lastly, we collected our samples from various classes and instructors. Student perceptions of the course workload, teaching quality, and fairness of assessment influenced student choices of learning strategies (Lizzio et al., 2002). It is, therefore, possible that other external factors could influence computer programming learning strategies. All in all, we plan to further investigate this phenomenon using data obtained from junior/senior-level undergraduate students.

5. CONCLUSIONS

The most significant contribution of this work is the development, testing and validation of the Deep Learning in Computer Programming (DL-CP) and Shallow Learning in Computer Programming (SL-CP) scales. This work lays the groundwork for further research into the intersection of coding grit, coding self-efficacy and student learning strategy selection in programming courses. The goal of this work is to better understand why some students struggle in programming courses and to equip instructors with the knowledge needed to help these students succeed.

Despite IT researchers' long tradition of modifying scales to fit specific computer-related tasks, previous work in this area has often utilized generic scales, which may fail to capture the important differences between computer programming courses and other IT or general education courses. By creating coding specific scales for deep and shallow learning strategies, this work also provides tools that others investigating the student achievement in computer programming courses may use to better understand the antecedents of student success or failure.

Future work should examine the relationships between coding grit, coding self-efficacy, shallow and deep learning strategies, and student outcomes in both introductory and advanced programming courses. A longitudinal study of how the learning strategies change with increased

programming skills may provide pedagogical insights to instructional scaffolding. It may also be fruitful to explore whether student cognitive learning strategies are associated with learning goals and persistence in computer-related majors.

6. REFERENCES

Anderman, E.M. (1992). Motivation and Cognitive Strategy Use in Reading and Writing. In the annual meeting of the National Reading Conference, San Antonio, Texas.

Beattie IV, V., Collins, B., & McInnes, B. (1997). Deep and surface learning: a simple or simplistic dichotomy?. *Accounting Education*, *6*(1), 1-12.

Bandura, A. (1977). Self-efficacy: toward a unifying theory of behavioral change. *Psychological Review*, *84*(2), 191.

Bandura, A. (1997). Efficacy: The Exercise of Control. *Freeman, New York.*

Barrington, K., & Johnson, D. (2005). The relationship between lab attendance and academic performance in a computer information systems course. In Proceedings of ISECON 2005, Vol. 22 (Columbus OH): 3573

Boyle, E. A., Duffy, T., & Dunleavy, K. (2003). Learning styles and academic outcome: The validity and utility of Vermunt's Inventory of Learning Styles in a British higher education setting. *British Journal of Educational Psychology*, *73*(2), 267-290.

Cano, F. (2005). Epistemological beliefs and approaches to learning: Their change through secondary school and their influence on academic performance. *British Journal of Educational Psychology*, 75(2), 203−221.

Compeau, D. R., & Higgins, C. A. (1995). Computer self-efficacy: Development of a measure and initial test. *MIS Quarterly*, 19(2), 189-211.

Compeau, D., Gravill, J., Haggerty, N., & Kelley, H. (2006). Computer self-efficacy. In P. Zhang and D.F. Galletta (Eds.), *Human-computer Interaction and Management Information Systems: Foundations, Advances in Management Information Systems, 225-261.*

Duckworth, A.L., & Quinn, P. D. (2009). Development and Validation of Short Grit Scale (Grit-S). *Journal of Personality Assessment,* 91(2), 166-174.

Elliot, A.J., McGregor, H.A., & Gable, S. (1999). Achievement goals, study strategies, and exam performances: A mediational analysis. *Journal of Educational Psychology*, 91(3), 549−563.

Fenollar, P., Roman, S., & Cuestas, P.J. (2007). University students' academic performance: An integrative conceptual framework and empirical analysis. *British Journal of Educational Psychology*, 77(4), 873−891.

Jones, C. H. (1984). Interaction of absences and grades in a college course. *The Journal of Psychology*, 116(1), 133-136.

Lavin, D.E. (1956). The Prediction of Academic Performance, Russell Sage Foundation, New York.

Lin, G-Y. (2006). Self-efficacy beliefs and their sources in undergraduate computing disciplines: an examination of gender and persistence. *Journal of Educational Computing Research*, 53(4), 540-561.

Lister, R. (2011). Computing education research geek genes and bimodal grades. *ACM Inroads*, *1*(3), 16-17.

Lizzio, A., Wilson, K., & Simons, R. (2002). University students' perceptions of the learning environment and academic outcomes: implications for theory and practice. *Studies in Higher education*, *27*(1), 27-52.

Lorenzen, T., & Chang, H. L. (2006). MasterMind©: a predictor of computer programming aptitude. *ACM SIGCSE Bulletin*, *38*(2), 69-71.

Mahatanankoon, P. (2018). Exploring the Antecedents to Computer Programming Self-Efficacy. In *Proceedings of the 10th International Conference on Advances in Information Technology* (pp. 1-6).

Mahatanankoon, P., & Sikolia, D. W. (2017). Intention to Remain in a Computing Program: Exploring the Role of Passion and Grit. In the 23rd Americans Conference on Information Systems (AMCIS), Boston (pp. 1-10).

Miller, R., Greene, B., Montalvo, G., Ravindran, B., & Nicholls, J. (1996). Engagement in academic work: The role of learning goals, future consequences, pleasing others, and perceived ability. *Contemporary Educational Psychology*, 21(4), 388−422.

Marton, F., & Säljö, R. (1976a). On qualitative differences in learning: I—Outcome and

process. *British Journal of Educational Psychology*, *46*(1), 4-11.

Marton, F., & Säljö, R. (1976b). On qualitative differences in learning—ii Outcome as a function of the learner's conception of the task. *British Journal of Educational Psychology*, 46(2), 115-127.

McClendon, C., Neugebauer, R. M., & King, A. (2017). Grit, Growth Mindset, and Deliberate Practice in Online Learning. *Journal of Instructional Research*, *8*, 8-17.

McGettrick, A., Boyle, R., Ibbett, R., Lloyd, J., Lovegrove, G., & Mander, K. (2005). Grand challenges in computing: Education—a summary. *The Computer Journal*, *48*(1), 42-48.

Murphy, S. M., & Tyler, S. (2005). The relationship between learning approaches to part-time study of management courses and transfer of learning to the workplace. *Educational Psychology*, *25*(5), 455-469.

Pea, R. D., & Kurland, D. M. (1984). On the cognitive effects of learning computer programming. *New ideas in psychology*, *2*(2), 137-168.

Rex, K., & Roth, R. M. (1998). The relationship of computer experience and computer self-efficacy to performance in introductory computer literacy courses. *Journal of research on computing in education*, *31*(1), 14-24.

Robins, A. (2010). Learning edge momentum: A new account of outcomes in CS1. *Computer Science Education*, *20*(1), 37-71.

Strayhorn, T. L. (2014). What role does grit play in the academic success of black male collegians at predominantly white institutions?. *Journal of African American Studies*, *18*(1), 1-10.

Watkins, D. (2001). *Correlates of approaches to learning: A cross-cultural meta-analysis.* In R. J. Sternberg & L.-f. Zhang (Eds.), *The educational psychology series. Perspectives on thinking, learning, and cognitive styles* (p. 165−195). Lawrence Erlbaum Associates Publishers.

Wolf, J. R., & Jia, R. (2015). The role of grit in predicting student performance in introductory programming courses: an exploratory study. In the 2015 *SAIS 2015 Proceedings*, *21*. Retrieved online from http://aisel.aisnet.org/sais2015/21

Editor's Note:

This paper was selected for inclusion in the journal as an EDSIGCON 2020 Distinguished Paper. The acceptance rate is typically 7% for this category of paper based on blind reviews from six or more peers including three or more former best papers authors who did not submit a paper in 2020.

Appendix A

DVs	Sample 1					Sample 2				
	IDV	Est.	t	p	95% CI L/U	IDV	Est.	t	p	95% CI L/U
Coding SE (C-SE)	SL-CP	0.148	0.935	.354	-0.17/0.47	SL-CP	0.458	2.461	.018	0.08/0.83
	DL-CP	-0.222	-1.413	.164	-0.54/0.09	DL-CP	-0.606	-3.286	.002	-0.98/-0.23
	DL-A	0.045	0.279	.781	-0.28/0.37	DL-A	-0.026	-0.120	.905	-0.46/0.41
						DL-D	0.219	1.151	.256	-0.16/0.60
	F=.794, p-value=.503, R^2=.046					F=3.448, p-value=.016, R^2 =.243				
Coding Grit (C-G)	SL-CP	-0.048	-0.512	.611	-2.84/1.69	SL-CP	0.193	2.327	.025	0.31/4.33
	DL-CP	-0.076	-0.819	.416	-3.26/1.33	DL-CP	-0.176	-2.131	.039	-4.10/-0.11
	DL-A	-0.022	-0.230	.819	-2.58/2.05	DL-A	0.061	0.643	.524	-1.58/3.05
						DL-D	0.005	0.067	.947	-1.99/2.12
	F = .657, p-value = .582, R^2 = .038					F = 1.866, p-value=.1338, R^2=.148				

Table 7 Regression Results

Shallow Learning in Computer Programming (SL-CP)

SL1: I try to memorize the steps for solving programming problems presented in the text or in the lecture.
SL2: When I study for the tests I review my class notes and look at solved programming problems.
SL3: When I study for tests I used solved programming problems in my notes or in the book to help me memorize the 'programming' steps involved.
SL4: I find reviewing previously solved programming problems to be a good way to study for a test.
SL5: In order for me to understand what technical terms meant, I memorized the textbook definitions.

Deep Learning in Computer Programming (DL-CP)

DL1: When studying, I try to combine different pieces of information from course material in new ways.
DL2: I draw pictures or diagrams to help me solve some programming problems.
DL3: I work on several programming examples of the same type of problems when studying this class so I can understand the problems better.
DL4: I work practice programming problems to check my understanding of new concepts or rules.
DL5: I examine example programming problems that have already been worked to help me figure out how to do similar 'coding' problems on my own.
DL6: I classify programming problems into categories before I begin to work them.
DL7: When I work a programming problem, I analyze it to see if there is more than one way to get the right solution.
DL8: While learning new programming concepts, I try to think of practical applications.
DL9: I put together programming ideas or concepts and draw conclusions that were not directly stated in course materials.
DL10: I work on practice programming questions/problems to check my understanding of new concepts or rules.
DL11: When I finish my programming practice questions/problems I check my solution for syntax errors.

Additional Survey Items for Pilot Sample 2

D1: Some programming problems can be visualized using diagrams and models.
D2: I develop models or pictures to help me visualize how programming work.
D3: I model different program modules or functions using some diagramming techniques.
D4: I use some diagramming techniques to understand how programming work.