Journal of Information Systems Education

Volume 36
Issue 4
Fall 2025

Development of the Cognitive Programming Engagement (CPE) Scale: A Learning Strategy

Pruthikrai Mahatanankoon and James R. Wolf

Recommended Citation: Mahatanankoon, P., & Wolf, J. R. (2025). Development of the Cognitive Programming Engagement (CPE) Scale: A Learning Strategy. *Journal of Information Systems Education*, 36(4), 400-416. https://doi.org/10.62273/KJKC8408

Article Link: https://jise.org/Volume36/n4/JISE2025v36n4pp400-416.html

Received:	November 15, 2024
First Decision:	February 10, 2025
Accepted:	April 27, 2025
Published:	December 15, 2025

Find archived papers, submission instructions, terms of use, and much more at the JISE website: https://jise.org

ISSN: 2574-3872 (Online) 1055-3096 (Print)

Journal of Information Systems Education, 36(4), 400-416, Fall 2025
https://doi.org/10.62273/KJKC8408

Development of the Cognitive Programming Engagement (CPE) Scale: A Learning Strategy

Pruthikrai Mahatanankoon
James R. Wolf
School of Information Technology
Illinois State University
Normal, IL 61790, USA
pmahata@ilstu.edu, jrwolf@ilstu.edu

ABSTRACT

Advances in information and communication technologies (ICT) coupled with artificial intelligence have made computer programming skills indispensable for IT majors and for an increasing number of other science, technology, engineering, and mathematics (STEM) disciplines. Like any hands-on skill, mastering computer programming requires dedicated time, patience, focus, and persistent effort. Understanding students' learning strategies as they engage in computer programming activities can reduce attrition and lay a solid foundation for a successful career in IT/STEM disciplines. This paper focuses on developing the cognitive programming engagement (CPE) scale, which builds on existing cognitive engagement measures. Self-reported data from undergraduate IT students who are learning computer programming show that CPE supports four-dimensional learning strategies: memorization, practice, analysis, and visualization, which aligns with the levels of Bloom's Taxonomy. The new scale supports confirmatory, discriminant, and predictive validity and tests on programming self-efficacy and coding grit with acceptable predictive validity. IT/STEM educators can use the scale to assess and evaluate students' learning and improve their teaching strategies.

Keywords: Bloom's taxonomy, Cognitive engagement, Computing education, Learning styles, Pedagogy, Programming

1. INTRODUCTION

Introductory programming courses serve as gateways to careers in information technology (IT) and an increasing number of science, technology, engineering, and math (STEM) fields. However, many IT educators can attest that it is extremely challenging to teach an introductory programming course. Practical teaching tips and pedagogical approaches to improve learning are frequently shared among fellow instructors (Menon, 2023; Zhang et al., 2020). Unfortunately, notoriously high failure rates (Bennedsen & Caspersen, 2019) and disheartening student experiences lead many students to quit or switch majors (Giannakos et al., 2017; Obaido et al., 2023).

Research suggests that low levels of student engagement may contribute to challenges in programming classes (Morgan et al., 2018a; Morgan et al., 2018b). IT students typically enter college with strong high school GPAs and SAT scores—two measures often associated with collegiate success—near the top of all college majors (College Board, n.d.; Westrick et al., 2021). Yet, IT majors score near the bottom of all students in terms of student engagement (Butler et al., 2016; Morgan et al., 2018a; Morgan et al., 2018b). As a result, some have suggested that improving student engagement might be the key to improving outcomes and reducing attrition in computing majors (Morgan et al., 2018a; Morgan et al., 2018b).

However, many computing educators are concerned that existing student engagement scales, such as the National Survey of Student Engagement (NSSE), do not accurately capture the engagement crucial for success in programming courses (Butler et al., 2016; Sinclair et al., 2015). These educators suggest that the problem is not with the students but with the scales used to capture student engagement. Supporting this, fourth-year computing students, who have already successfully navigated introductory programming courses, score lower on student engagement than first-year computing students (Sinclair et al., 2015). Many computing academics believe that scales focusing on metrics such as research paper length or reflective learning offer little insight into the domain-specific cognitive and metacognitive skills essential for programming success (Butler et al., 2016; Morgan et al., 2018a; Morgan et al., 2018b). We agree and believe that a new, tailored instrument is needed to assess domain-specific skills in programming education.

Student engagement is a multifaceted construct composed of three distinct subconstructs: cognitive engagement, behavioral engagement, and emotional engagement (Fredricks & McColskey, 2012). In this study, we focus exclusively on the cognitive engagement subscale as it pertains to computer programming. This work addresses a gap in the current literature by developing and validating the cognitive programming engagement (CPE) scale. The CPE scale builds on existing cognitive engagement measures and computer learning strategies (Greene, 2015; Mahatanankoon & Wolf, 2021). Ultimately, this work aims to empower computing faculty with the knowledge and tools to improve student success, retention, and diversity in computing courses, as well

as providing learners with a framework to reflect on their learning.

2. COGNITIVE ENGAGEMENT AND COMPUTER PROGRAMMING

Student engagement is the student's psychological commitment and purposeful dedication to acquire, comprehend, or excel in terms of knowledge, skill, or craft (Newman et al., 1992). It is positively associated with academic performance, retention, and graduation and is one of the most widely studied educational constructs (Fredricks & McColskey, 2012; Yu et al., 2021). Student engagement has three subconstructs: cognitive, behavioral, and emotional (Wong & Liem, 2022). Other proposed subconstructs include behavioral, psychological, socio-cultural, and holistic (Kahu, 2013). In contrast to generic student engagement measures that focus on time spent or effort expended, cognitive engagement embodies the student's purposeful and strategic investment in the learning process (Greene, 2015).

Further, student engagement focuses on two components: one related to the students and the other centered on the institution (Kuh, 2001; Wolf-Wendel et al., 2009). The student-centric aspect involves students' time and effort in their studies and the various activities contributing to their academic success (Kuh, 2001). The organization-centric aspect pertains to how higher education institutions allocate resources and structure learning opportunities to encourage student participation and benefit from such activities (Kuh, 2001). In computing studies, the bulk of student engagement research focuses on instructor-centric aspects or instructional innovation (Gunness et al., 2023; Hazzam & Wilkins, 2023; Hsiao et al., 2022). In these studies, researchers implement an instructional innovation or organizational change and then test for variations in student engagement. For example, research found that perceived teaching presence impacted online students' engagement (Zhang et al., 2016).

As behavioral engagement is the easiest to measure, the bulk of earlier computing studies on instructional innovation focused on students' behavioral engagement (Davies, 2002). Likewise, the majority of items in existing student engagement scales, such as the National Survey of Student Engagement (NSSE), also capture student behaviors (Butler et al., 2016). However, a growing number of recent computing-related student engagement studies have explored cognitive engagement (Gunness et al., 2023; Zhong, 2023).

Behavioral and cognitive engagement are closely related but distinct subconstructs. Behavioral engagement refers to a student's active participation and positive conduct in both academic and school-related activities (Fredricks et al., 2004). In contrast, cognitive engagement involves students' mental investment in their learning, their commitment to learning, strategic thinking, and readiness to invest effort (Fredricks et al., 2004). Items such as class participation and valuing school can span both behavioral and cognitive engagement.

This work focuses on a domain-specific learning strategy of cognitive engagement within the context of computer programming. This focus is crucial because the existing student engagement measures may not capture the unique demands of programming education.

2.1 Cognitive Engagement: Shallow and Deep Learning

Cognitive engagement encapsulates a student's deliberate endeavors to comprehend new concepts and cultivate new skills (Greene, 2015). It goes beyond passive information absorption and involves actively seeking understanding, developing new skills, and constructing meaningful connections between ideas (Greene, 2015). The original cognitive engagement scale has three sub-scales: self-regulation, deep strategy use, and shallow strategy use. According to Greene (2015), the shallow and deep strategy use sub-scales are two distinct "learning strategies."

Deep cognitive engagement connects novel concepts with existing knowledge, intentional practice, and elaboration. Elaborative processing is so closely intertwined with deep engagement that these terms are often used interchangeably in the literature (Greene, 2015). Effective learning strategies such as flashcards, self-quizzing, and spaced studying are all linked to deep cognitive engagement (Greene, 2015). In contrast, shallow engagement emphasizes memorizing facts and details and focuses on finishing learning tasks rather than obtaining understanding (Li & Lajoie, 2022). Shallow cognitive engagement is most associated with cramming and rote memorization (Dunlosky et al., 2013). Cramming improves factual recall for immediate assessments but does not cultivate the deeper understanding, critical thinking skills, or long-term knowledge retention needed for academic success (Kornell, 2009; Rawson & Kintsch, 2005).

However, just as cramming does work in specific settings, shallow learning techniques may also be a successful strategy for some academic tasks. Deep learning strategies have long been associated with success, whereas shallow cognitive processing, or shallow learning strategies, has received less favorable attention (Asikainen & Gijbels, 2017). Recent research has revealed a more nuanced perspective. Notably, accomplished students employ deep and shallow techniques, tailoring their approach to specific academic tasks (Mahatanankoon & Wolf, 2021).

2.2 Deep and Shallow Learning Strategies in Computer Programming

Emerging research suggests that college students may employ a mixture of deep and shallow processing strategies depending on the specific learning task and context, which further complicates the traditional "deep good, shallow bad" dichotomy (Asikainen & Gijbels, 2017; Greene, 2015). Studies assessing the evolution of college students' learning strategies over time yielded mixed results. These mixed results suggest that learning strategies are far more intricate than a simple binary of deep being effective and shallow being less so (Vrugt & Oort, 2008).

Supporting this, research has emphasized the need for domain-specific considerations when measuring cognitive engagement and cited previous challenges in differentiating deep and shallow engagement in mathematics (Greene, 2015). Mahatanankoon and Wolf (2021) posited that shallow learning strategies might be more effective in introductory classes. In contrast, deep learning strategies become more relevant in advanced courses, which suggests potential variations in optimal deep-learning strategies across different IT specializations. Their findings align with cognitive theory (Chi et al., 1981) and emphasize the importance of mastering deep and shallow cognitive processes for academic success.

2.3 The Need for a New Cognitive Engagement Scale for Computer Programming

Recognizing the importance of programming competence for computing students' career success, both computer science (CS) and information systems (IS) professional associations and accrediting agencies heavily emphasize programming instruction in their accreditation processes. In their model curriculum, IS 2020, the joint task force of the Association for Computing Machinery (ACM) and the Association for Information Systems (AIS) identified application development and programming as two required competencies for graduates of undergraduate programs in IS (Leidig & Anderson, 2020). Similarly, the Accreditation Board for Engineering and Technology (ABET) mandates extensive programming and software development coverage within its computer science and similarly named computing program criteria (ABET, 2025).

While there is near universal acceptance that student engagement is beneficial for learning, there is an open debate—especially among computing educators—on the appropriateness of current instruments used to measure student engagement. IT educators measure student engagement based on the emotional-cognitive dimensions (i.e., vigor, dedication, and absorption) of task-specific activities (Schwarz & Zhu, 2015). Nonetheless, IT students consistently demonstrate lower engagement levels in various international benchmark surveys, including the National Survey of Student Engagement (NSSE), the Student Experience Survey (SES), and the United Kingdom Engagement Survey (UKES; Morgan et al., 2018a; Morgan et al., 2018b). The SES and UKES were designed to assess students' educational experience and engagement in Australian and UK higher education institutions (Morgan et al., 2018a; Morgan et al., 2018b).

The cognitive processes and skills needed to learn computer programming are unique and typically not required in other learning domains (Renumol et al., 2010). Supporting this, Silva et al. (2024) found that students who employed programming-specific learning strategies were more likely to be successful in their programming courses. Further, high-performing students were more likely to use programming-specific strategies than low-performing students (Silva et al., 2024). Recognizing this, IT educators have raised concerns about the suitability of these widely used student engagement instruments for computing students. Many suggest that these measures may not capture relevant aspects of engagement or use terminology that students may interpret differently (Morgan et al., 2018a; Morgan et al., 2018b; Sinclair et al., 2015). These educators suggest that current student engagement scales fail to capture the unique engagement activities required for success in programming courses and computing degree programs.

This failure warrants the development of a new instrument specifically tailored to the unique demands of programming education. While valuable in other contexts, student engagement scales that focus on metrics such as paper length (Kuh, 2001) offer limited insight into the domain-specific cognitive and metacognitive skills essential for programming success (Zhang et al., 2020). By identifying and addressing these limitations, a domain-specific engagement measure can provide valuable insights into student learning strategies and inform the development of effective interventions to improve student outcomes in programming education.

Further, an active community of computing educators is involved in student engagement research (Bai et al., 2021; Lai, 2021; Saqr et al., 2023). However, many of these studies use student engagement scales designed for more general educational contexts. These generic scales, while appropriate for many academic disciplines, may fail to capture the nuances of engagement crucial for success in computing courses. Previous research also advocates sub-dimensions of cognitive engagement in computer programming learning environments (Mahatanankoon & Wolf, 2021).

Fortunately, IT has a rich history of developing domain-specific constructs tailored to explicitly capture IT-related attitudes and behaviors, with established tools like computer self-efficacy (Compeau & Higgins, 1995), IT adoption based on the theory of planned behavior (Mathieson, 1991), and the many incarnations of the technology acceptance model (TAM; Davis, 1989) providing a roadmap for scale development (Straub et al., 2004). This work seeks to develop and validate a cognitive engagement assessment tool designed explicitly for programming education.

3. METHOD

3.1 Existing Cognitive Engagement Constructs

The first step in developing the CPE scale was to examine the existing literature. Greene's (2015) excellent cognitive engagement measures are the "gold standard" in educational psychology. Fueled by intrinsic motivation and self-regulation, *cognitive engagement* stresses the *mental* exertions, not behavioral efforts, to learn new skills (Fredricks et al., 2004); cognitive engagement is defined as "thoughtfulness and willingness to exert the effort necessary to comprehend complex ideas and master difficult skills" (p. 60). While the construct is geared toward a generic educational setting, Greene (2015) emphasized the need for domain-specific considerations when measuring cognitive engagement and later appended self-regulation and persistence as additional sub-dimensions. Miller et al. (1996) provided an example of how to adapt the scales for use in a mathematics course. This study utilizes the two original cognitive engagement sub-scales (Fredricks et al., 2004; Greene, 2015; Miller et al., 1996): deep and shallow learning (processing) strategies. Based on the literature and the preliminary work by Mahatanankoon and Wolf (2021), we propose the 21-item CPE scale (see Appendix A).

3.2 Procedure and Sample

We conducted our scale development in four steps (Igbaria & Baroudi, 1993; Pett et al., 2003; Straub et al., 2004): 1) pre-analysis, 2) extracting the initial factors, 3) evaluating and refining the exploratory factors, and 4) examining construct validity through confirmatory factor analysis.

We developed the CPE scale using data collected from undergraduate IT students (i.e., computer science, cyber-security, information systems, and network telecommunications majors) enrolled in our ABET-accredited institution in the Midwest region of the United States. Data collection, assisted by introductory computer programming course instructors, occurred from fall 2020 to spring 2022. The identified instructors forwarded the web-based survey links to their students. The survey was voluntary, but some instructors offered extra credits for participants.

We received 261 completed responses (fall 2020, n=28; spring 2021, n=138; fall 2021, n=27; spring 2022, n=68). We dropped incomplete responses (n=49), non-IT majors (n=44),

and graduate students (n=6). The incomplete responses were excluded because the dependent variables (CPE items) were missing (Hair et al., 1995). Responses that were completed in non-English browsers were also eliminated (n=8). Our data had 120 IT students who enrolled in introductory (IT 1xx) programming classes (i.e., Java and C++) from fall 2020 to spring 2022. See Table 1 below. The sample comprised 85.8% men (n=103), 12.5% women (n=15), and two unknowns. The IT majors included 50.8% computer science (CS; n=61), 26.7% cyber-security (CyS; n=32), 18.3% information systems (IS; n=22), and 4.2% network telecommunications (NT; n=5). Independent t-tests showed no significant difference in research variables between men and women, except for SL5 (t=1.78, df=21.39, p=.045), suggesting that both groups provided similar answers to the questionnaire. There was no significant age difference between the two genders (t=.447, df=116, p=.656).

Course Title	Major (CS, IS, CyS, NT)	Gender (M, F, U)
Structured Problem Solving	(9, 3, 6, 0)	(16, 2, 0)
Scripting Languages	(0, 1, 13, 3)	(14, 1, 2)
Application Programming	(2, 16, 10, 2)	(27, 3, 0)
Data Structures	(19, 1, 2, 0)	(18, 4, 0)
C++ Programming	(31, 1, 1, 0)	(28, 5, 0)
Chi-square	χ^2=102.67, df=8, p<001	χ^2=13.79, df=8, p=.087

Table 1. Sample's Characteristics

3.2.1 Step 1: Pre-Analyses of Correlation Matrix. An initial examination of the 21x21 correlation matrix (SL1-SL5 and DL1-DL16) suggested that the shallow learning items were significantly correlated, while several items from deep learning were also correlated among themselves (DL1-DL16). Bartlett's Test of Sphericity, a test of "singular correlation matrix," suggested that factor analysis was feasible (chi-square=1163.11, df=210, p<.001; not an identity matrix; Pett et al., 2003). Our correlation matrix yielded a Kaiser-Meyer-Olkin (KMO) value of .765 (>.60), which suggested a sufficient sample size relative to the number of items in our proposed scale (Pett et al., 2003). The measure of sampling adequacy (MSA) statistics revealed weak correlations of SL4 (.574) and DL11 (.565) among other items. Appendix C shows the correlation matrix of all items.

3.2.2 Step 2: Extracting Initial Factors and Rotation. We ran unrotated principal component analysis (PCA) using SPSS v.28 to extract the initial factors, which yielded a six-factor solution. PCA is suitable for exploratory factor analysis (Pett et al., 2003). Initial inspection of the factor loadings showed that shallow cognitive engagement (SL1-SL4) loaded strongly onto the same component, except for SL5 (.442). Since SL5 had low correlation values among other shallow cognitive engagements (see Appendix C), it was dropped from further analysis. We scrutinized DL6 ("*I classify programming problems into categories before I begin to work them*") as it correlated with the shallow cognitive engagement items. Given its low communality value (.351) and the ambiguity of "classify," DL6 was eliminated from our factor analysis because of a community value of less than .40.

After eliminating the previously mentioned items, we performed PCA with orthogonal varimax rotation to help interpret the meaning of the factors (Igbaria & Baroudi, 1993). We eliminated items with cross-loadings (>.40) and loadings with less than .50, given our sample size needed for significance (Hair et al., 1995). DL1 ("*When studying, I try to combine different pieces of information from the course material in new ways*") suffered from cross-loadings, while SL5 ("*In order for me to understand what technical terms meant, I memorized the textbook definition*") had insufficient factor loading with a communality value of less than a .40. 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*") also did not fall into any dimensions. These items were eliminated from further analyses.

3.2.3 Step 3: Evaluating and Interpreting the Factors. The remaining items produced a five-factor solution with an eigenvalue above 1.0. Eigenvalues indicate the total variance accounted for by each factor (Hair et al., 1995). The five-factor solution accounted for nearly 73% of the total variance. While previous studies demonstrated that shallow and deep cognitive engagements are two distinct constructs (Fredricks et al., 2004; Greene, 2015; Miller et al., 1996), our results reveal the existence of sub-dimensions beyond shallow learning in the context of computer programming. Appendix D shows each factor's internal composite reliability (Cronbach's alpha) and the maximum internal composite reliability for each factor with deleted items (if fewer questions were possible). Only SL1 and DL7 showed a negligible increment of alpha values when deleted.

3.2.4 Step 4: Confirmatory Factor Analysis. We reassessed our factors using confirmatory factor analysis (CFA). CFA is often used in scale development and construct validation. Since we already examined the factor loadings from the previous steps, CFA can be used to assess each item with "fewer parameter estimates than exploratory factor analysis" (Brown, 2015, p. 37). By fixing cross-loading among items to zero, standardized factor loadings and model-fit indices foster more "parsimonious" measurement models (Brown, 2015). We used R-Studio with lavaan statistical package to estimate each item's standardized loading and the measurement model's fit indices, i.e., CFI (>.95), TFI (>.95), and RMSEA (<.05). Fit indices captured how well our exploratory results of factors (Appendix D, "default model") fit between the independence model ("worse model," fit indices=0) and the "saturated model" (or "just identified model," fit indices=1). Our model produced CFI=.887, TLI=.859, and RMSEA=.090. In reviewing each standardized loading, we found that SL1 and DL11 explained 35.2% and 34.5% of factor 1 and factor 5, respectively. These two items resided on the lower bound of reliability (Harrington, 2009). DL16 also had a standardized loading of 1.07 (a negative residual variance). After eliminating these items (i.e., DL1, DL11, and DL16), our model produced CFI=.906, TLI=.869, RMSEA=0.093. The values point to a moderate-acceptable fit of a four-factor solution (Harrington, 2009). Given the small size, researchers should be cautious about the interpretation of

our model. Based on our final CFA results, we labeled our CPE as follows: shallow, visual, practical, and analytical dimensions. See Appendix B for details.

We demonstrated convergent and discriminant validity using suggestions from Straub et al. (2004). Convergent validity can be demonstrated by observing a higher than 0.5 of the standardized loading estimates along with a composite reliability (CR) value exceeding 0.7 (Hair et al., 1995). However, excluding SL1 and DL11 was problematic because of the different "acceptable" standardized loadings (Harrington, 2009; Straub et al., 2004). Convergent validity is demonstrated in Appendix E, showing that once the irrelevant items had been eliminated, the retained items loaded "cleanly" on their designated construct and did not load onto another construct (Straub et al., 2004).

While convergent validity establishes inter-item relationships within a component (factor), discriminant validity is "the degree to which items differentiate between constructs or measure distinct concepts" (Igbaria & Baroudi, 1993, p. 142). We assessed discriminant validity by examining the variance shared between the items (squared correlations), which should be lower than the average variance extracted (AVE) of the items loaded onto a factor (Igbaria & Baroudi, 1993). All CPE dimensions satisfy discriminant validity criteria (see Appendix E). Shallow CPE, bearing no correlation with any "deep" CPEs, is empirical evidence that the proposed measures will support the theoretical cognitive engagement constructs.

4. DIMENSIONS OF COMPUTER COGNITIVE ENGAGEMENT (CPE)

Our analyses demonstrate that cognitive engagement has four unique dimensions related to computer programming.

4.1 Dimension 1: Shallow-CPE

Shallow-CPE (cpe-S) is *a coding strategy to memorize programming syntax, coding patterns, or steps*. This fundamental cognitive engagement occurs when novice programmers can memorize and understand the syntax of a programming language. It maps to remembering and understanding in Bloom's Taxonomy. Shallow-CPE coincides with rote learning (Mayer, 2002). During this learning stage, programming students memorize the steps or patterns needed to solve a given programming problem. However, new programming problems pose a challenge for this level of cognitive engagement. Similar to the different levels of Bloom's Taxonomy—lower learning levels support higher learning levels—Shallow-CPE provides a solid foundation for other advanced CPEs. This dimension aligns with the previous empirical studies (Greene, 2015; Miller et al., 1996). The items related to Shallow-CPE are:

- When I study for the tests, I review my class notes and look at solved programming problems (SL2).
- When I study for tests, I use 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).

4.2 Dimension 2: Practical-CPE

Practical-CPE (cpe-P) is defined as *a coding strategy focused on task repetition of exercises to form new programming concepts and skills*. The strategy involves doing by repetition, and through these reiterations, the programmers may start to notice recurring new coding patterns or new ways to solve programming problems. Therefore, this cognitive engagement dimension requires that programmers continuously work on coding assignments and examples. Practical-CPE involves iterative hands-on experiences required to realize this level of cognitive engagement. When programming instructors advocate that their students must engage in hands-on practices and exercises, they are intentionally (or unintentionally) trying to cultivate Practical-CPE. Mayer (2002) suggested that this level of learning is comprised of two cognitive activities, "closely linked to procedural knowledge": *executing*—doing exercises familiar to learners; and *implementing*—applying learned procedures to unfamiliar tasks. This level closely aligns with Bloom's Applying the learned programming skills. The items related to Practical-CPE are:

- 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 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).

4.3 Dimension 3: Analytical-CPE

Analytical-CPE (cpe-A) is defined *as a coding strategy seeking to form new programming logic and semantics*. It is closely aligned to Bloom's Analyzing—the ability to form new connections among different concepts. In this learning stage, new coding strategies along with new connections and semantics are formed. While Shallow-CPE helps form the syntaxes (grammar or representation), Analytical-CPE formulates the semantics (or language meaning). This dimension supports Mayer's definition of "meaningful learning"—the ability to transfer knowledge to new situations and "devising a way of achieving a goal that one has never previously achieved" (Mayer, 2002, p. 227). We believe that Analytical-CPE is the precursor to being a proficient computer programmer whose intention is to develop new user applications from scratch or to become a system software engineer. To novice programmers, Analytical-CPE is time-consuming and cognitively demanding. The Analytical-CPE dimension includes the following items:

- When I work on 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).
- I put together programming ideas or concepts and draw conclusions that were not directly stated in course materials (DL9).

4.4 Dimension 4: Visual-CPE

Visual-CPE (cpe-V) is defined as *a coding strategy that can be supplemented by modeling languages or diagramming tools*. When requirements are ambiguous, systems analysts and software engineers often exercise a variety of modeling

languages to depict software requirement specifications. Software requirements as well as documentation are captured through these standardized graphical models. Unified Modeling Language (UML, see www.uml.org), as well as other traditional modeling techniques, are good examples of these modeling tools.

Although proficient programmers may develop their own "concept maps" (Novak & Canas, 2008) to visualize the relationship between programming syntax and semantics, students without fundamental programming skills often struggle to create UML diagrams effectively. Understanding programming concepts is essential for accurately modeling system structures, behaviors, and interactions. This is why foundational programming courses are a prerequisite for Software Analysis and Design (SA&D) classes. These diagrams allow software designers to visualize the interrelated parts of hardware and software components. They also serve as "blueprints" for system behaviors or functionalities. When confronted with alternative software solutions, systems designers often develop multiple comparison models as a way to "evaluate" the efficiency, labor (programmer) cost, and maintainability of their software designs before commencing to program.

Visual-CPE operates at Bloom's Evaluating level, which involves synthesizing various components into a cohesive whole (Mayer, 2002). Visualizing software requirements demands additional effort beyond programming and often places a significant cognitive load on programmers, as they must align their design models with their functional programs. Effective programmers have a habit of documenting and creating visual representations of their computer programs for evaluation, modification, and maintenance. As such, the ability to develop diagrams and visualize how various code modules work extends beyond all previous CPEs. The items related to Visual-CPE are:

- I draw pictures or diagrams to help me solve some programming problems (DL2).
- Some "programming" problems can be visualized using diagrams or models (DL12).
- I develop models or pictures to help me visualize how programming works (DL13).
- I model different program modules or functions using some diagramming techniques (DL14).
- I use some diagramming techniques to understand how programming works (DL15).

In the next section, we will discuss the nomological validity of CPE dimensions. We elaborate on these dimensions later in the Discussion and Implications sections and offer practical suggestions based on our findings.

5. NOMOLOGICAL AND PREDICTIVE VALIDITY

To provide additional validity evidence of our measures, we tested the CPEs with other existing computer programming constructs: grit and self-efficacy. Predictive and nomological validity seek to establish relationships among the proposed measures and other extant constructs. Predictive validity demonstrates how an existing construct serves either as an independent or a dependent variable for the new measures, while nomological validity aims to test relationships among multiple constructs (Hagger et al., 2017).

5.1 Coding Grit

Individual differences in persistence and sustained effort influence academic achievement and overall life success (Duckworth, 2016). Among these factors, grit has emerged as a crucial construct that combines passion, long-term commitment, and unwavering dedication to achieving one's goals (Duckworth et al., 2007). Duckworth (2016) defined grit as "determination and direction" (p. 13). It reflects individuals' consistent pursuit of their passions despite encountering setbacks and challenges. Grit has a positive impact on learning outcomes regardless of instruction mode or subject matter (Pellas et al., 2024).

Individuals tend to develop higher grit as they grow older. This upward trend may be attributed to various factors, including self-awareness, exposure to diverse challenges, and the acquisition of effective coping strategies. Furthermore, grit can be strengthened through deliberate and focused practice, highlighting its potential for educational and professional intervention (McClendon et al., 2017). Wolf and Jia (2015) found that "general grit" predicts success in introductory programming courses and even exceeds college entrance exams in predictive power. While they explored the link between "general grit" and programming success, Mahatanankoon and Sikolia (2017) focused on "coding grit," a domain-specific concept tailored for programming. Their work established and validated "coding grit" as perseverance and focus within the programming domain. They also demonstrated a positive association between passion, coding grit, and positive attitudes toward programming, which suggested that fostering coding grit and passion might be key to promoting positive attitudes and ultimately retaining computer science majors (Mahatanankoon & Sikolia, 2017).

5.2 Coding Self-Efficacy

Computer self-efficacy, a well-established construct in information systems, is the belief in one's ability to use computers, and it is influenced by past experiences, observing others, encouragement, and even how individuals feel in specific situations (Compeau & Higgins, 1995). Compeau and Higgins (1995) adapted the broader concept of self-efficacy and Bandura's (1977) scale to the specific domain of computer use. Notably, self-efficacy is malleable and can be influenced by four key sources: performance accomplishments, vicarious experiences, verbal persuasion, and physiological states (Bandura, 1977).

Computer self-efficacy is the most-studied socio-cognitive attribute of beginning programming students and is positively associated with interest in computing, success in computer programming courses, and a desire to remain in computing majors (Clarke-Midura et al., 2019; Kanaparan et al., 2013). Computer self-efficacy is not fixed but can be improved (Compeau et al., 2007). Additionally, teaching methods employed in programming courses can influence students' computer self-efficacy (Tsai et al., 2023). This malleability suggests the potential for interventions aimed at enhancing computer self-efficacy. Building on Compeau and Higgins' (1995) work, the computer self-efficacy scale was adapted to measure IT-related self-efficacy. For example, coding self-efficacy, which pertains to one's belief in competence in computer programming, reveals a positive correlation between coding grit and programming self-efficacy (Mahatanankoon, 2018). As such, individuals with higher levels of perseverance

and passion for programming tend to exhibit greater confidence in their coding abilities. Figure 1 illustrates the overall constructs and proposes that coding grit influences different types of CPEs, which, in turn, affects coding self-efficacy.

5.3 Results

To demonstrate nomological validity, we correlated the sub-dimensions of CPEs with coding grit and coding self-efficacy. Table 2 reveals the correlations among the research variables in question.

Coding grit and self-efficacy were significantly correlated. The higher the programming grit, the higher one's coding self-efficacy. Coding grit was moderately correlated with Practical-CPE (cpe-P), while coding self-efficacy was significantly related to Analytic-CPE (cpe-A). The relationship between coding grit and Practical-CPE is reasonable since both constructs foster experiential learning. Moreover, Analytic-CPE may promote belief in one's programming skills—coding self-efficacy, or vice versa. From our nomological assumption, it was not a surprise that Shallow-CPE had no relationship with either coding grit or coding self-efficacy. These findings hypothetically supported the conceptual definitions of these constructs.

For predictive validity, coding grit was then hypothesized to predict CPE's sub-dimensions, which then influenced coding self-efficacy. Multiple regression analyses (MRA) were used to test the causal relationships among the variables. The average variance inflation factors (VIF) of regressing coding self-efficacy onto CPEs was 1.158, which showed that the multicollinearity of CPE dimensions was not a concern (see Section 3 for convergent and discriminant validity). Predictive validity tests revealed that coding grit predicted all deep-learning CPEs, but only practical CPE (cpe-P) yielded noticeable variance explanation. Coding grit was not a predictor of Shallow-CPE, which by itself also hindered coding self-efficacy. Practical-CPE (cpe-P) and Analytical-CPE (cpe-A) positively predicted coding self-efficacy. No causal relationship was found between visual CPE (cpe-V) and coding self-efficacy. Figure 2 illustrates our predictive validity results.

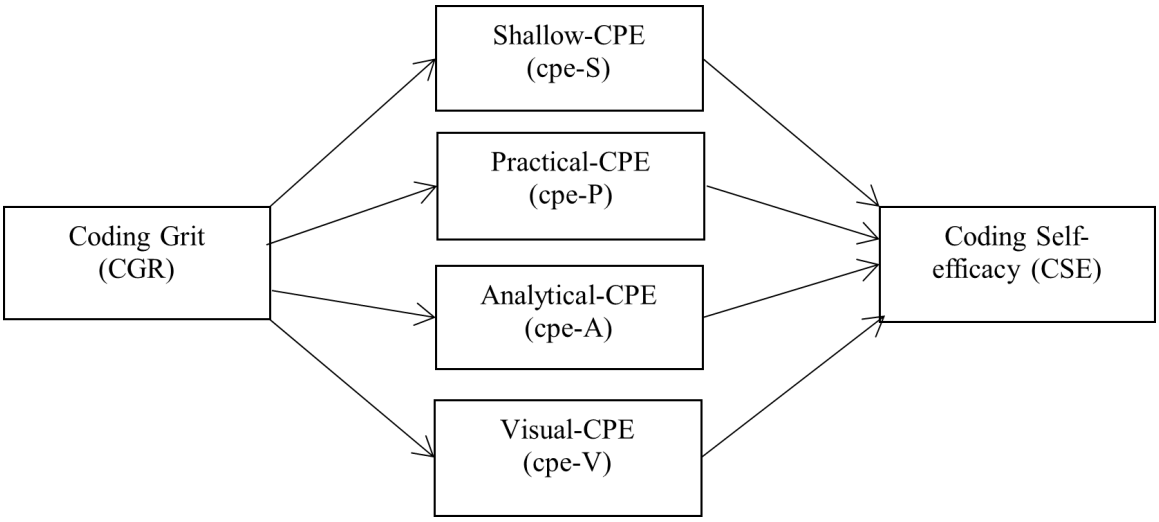


Figure 1. Nomological Network of Computer Programming Constructs

CPE	cpe-S	cpe-P	cpe-A	cpe-V	CGR	CSE
cpe-S	1	.144	.021	.100	-.168	-.143
cpe-P		1	.351**	.347**	.336**	.323**
cpe-A			1	.243**	.284**	.423**
cpe-V				1	.194*	.212*
CGR					1	.471**
CSE						1

(*= $p < .01$, **=$p < .001$)

Table 2. Correlation Matrix With Other Measures

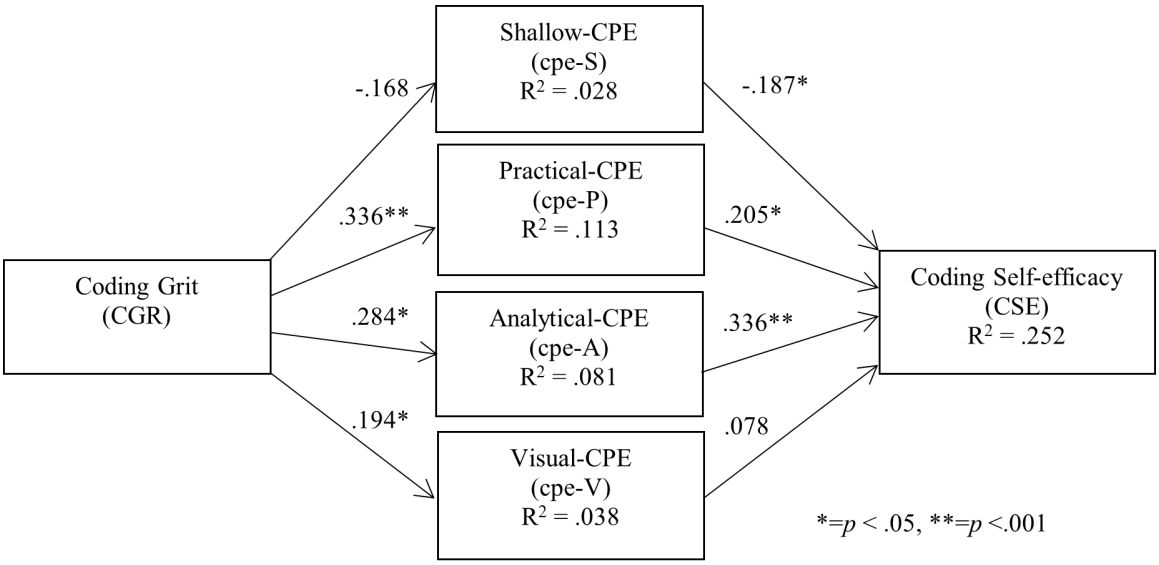


Figure 2. Predictive Validity

6. DISCUSSION AND IMPLICATIONS

Our nomological and predictive validity tests imply that: a) Shallow-CPE has less to do with hands-on coding and more with rote learning and memorizing. Shallow-CPE is the foundation to "remember" coding syntax, but it is an insufficient learning strategy to be a proficient programmer. Based on Bloom's Taxonomy (1956), Shallow-CPE operates at the foundational level of cognitive engagement of "remembering" and "understanding" programming language syntax and patterns. b) Practical-CPE focuses on hands-on experience, which is related to coding grit. Since grit constitutes the qualities of "continued interests" and "perseverance of efforts" (Duckworth, 2016), devoting time and energy to this cognitive engagement also influences one's coding self-efficacy. c) Coding self-efficacy is highly influenced by Analytical-CPE. Because coding self-efficacy is determined by grit and passion (Mahatanankoon, 2018; Mahatanankoon & Sikolia, 2017), this level of cognitive engagement corresponds to Bloom's Analyze level (i.e., finding new programming solutions, searching for practical use of learned syntax or patterns, and drawing new conclusions and ideas). d) With concentration on "modeling" and less engagement on active coding, Visual-CPE has a low significant relationship with coding grit and coding self-efficacy. However, the ability to model and visualize interrelated software components requires a higher level of cognitive engagement. We consider Visual-CPE to be aligned with at least Bloom's Analyze level, if not Evaluate. In some cases, system designers create a set of design diagrams to "evaluate" or "compare and contrast" the quality of software components and architectures, e.g., structured chart, design class diagram, and state machine diagram.

We, therefore, argue that CPE's four sub-dimensions align conceptually with Bloom's Taxonomy. CPE ranges from basic rote learning to practical and analytical learning, and to the ability to visualize programming logic. Figure 3 displays the mappings of CPE's sub-dimensions to Bloom's. Empirical evidence of CPE and its sub-dimensions has several implications, including the expansion of coding skills and grit as well as the development of IT artifacts/products.

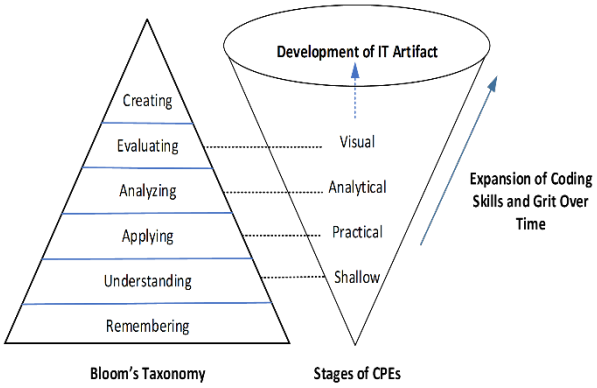


Figure 3. Aligning CPE Dimensions With Bloom's Taxonomy

6.1 Implications for Educators and Researchers

This work has several implications for educators and researchers. First, educators can use CPE to evaluate student learning and computer programming comprehension. Lorås et al. (2022) highlighted the importance of bridging the gap between general higher education theories and definitions and the specific research conducted in computing education. CPE offers a solution by providing computing educators with a domain-specific framework based on Greene's (2015) well-established theoretical work.

Next, changes in pedagogical approaches may advance students to higher CPE levels. For example, this work suggests that the first few weeks of an introductory programming class should focus on memorizing programming language syntax, including the three building blocks of a programming language (i.e., code sequencing, repetition (looping), and decision-

making). Once learners have internalized programming syntax and patterns, they should focus on the analytical or practical aspects of computer programming. This progression leads to higher levels of learning, such as applying their skills to solve new problems (Practical-CPE) or devising alternative solutions for existing problem sets (Analytical-CPE). In addition, CPEs may be used to support practical teaching techniques, such as "read before writing," "code early and often," and "yield results beyond teaching syntax" (Zhang et al., 2020, p. 115).

6.2 Students and Practitioners

This work has several implications for students and practitioners. First, practitioners and students may use this scale (see Appendix B) to guide their cognitive efforts and evaluate their programming skills and knowledge acquisition progress. Instead of waiting for formative and summative feedback from their instructors, IT students and beginning programmers can focus their study efforts via memorization, practice, analysis, and visualization. Armed with this knowledge, students can move from surface-level to deeper levels of cognitive engagement. Second, rote memorization learning strategies (i.e., cramming) are less effective for computer programming and may reduce long-term commitment to persist in their degree program. Our findings suggest that Shallow-CPE—a solid foundation for higher learning strategies—has a low correlation with the sub-dimensions of Deep-CPEs. Research suggests that shallow and deep learning strategies have an inverse relationship (Alexander, 2004). Succeeding in a programming course requires more than reviewing class notes and memorizing programming steps and solutions; it requires additional time and effort—not just to memorize but to "understand" how "things work." Deep CPEs (i.e., analytical, practical, and visual) will foster connections between programming syntaxes and semantics.

Finally, this work suggests that students are underutilizing diagrams and visualization tools. Visual-CPE is the least utilized cognitive learning strategy based on our data. Perhaps visualizing alone does not contribute to a final "executable" product or a specific programming outcome, similar to creating a *concept map* to connect complex ideas (Novak & Canas, 2008). In practice, visual diagrams "serve as an abstraction—an approximate representation of the real item that is being built," which allows programmers to see the systematic relationships or effects among various programs (Cernosek & Naiburg, 2004, p. 1). As such, Visual-CPE reveals significantly low correlations with coding grit and self-efficacy (see Table 2). Other measurable outcomes of Visual-CPE, such as effective coding, project success, code maintenance, or end-user acceptance, may be more appropriate to capture this learning strategy.

6.3 Limitations

We acknowledge well-known difficulties with self-reported data. However, we maintain that these instruments still hold value because they directly access students' perceptions of their motivation and engagement, which are crucial for understanding their learning processes and behaviors. Since an individual's contextual interpretation largely drives motivation, self-reporting offers unique insights into these subjective experiences. While all measurement methods have inherent limitations, self-reporting has historically contributed valuable findings to understanding motivation and engagement. Therefore, we believe that the solution to limitations with self-reported data is not to abandon these measures but to move beyond their exclusive use, augmenting self-reports with other methodologies, such as interviews, observations, and trace analyses, to achieve a more comprehensive and nuanced understanding of cognitive engagement. We plan to explore additional methodologies for capturing cognitive engagement in the programming domain in future work.

We advise educators to exercise caution when applying CPEs. First, our findings are based on our small sample (n=120) of convenience, with students passing their first introductory programming class. Nunnally (1978) suggested that at least 10 subjects per item are required; our sample size is nearly adequate for the final 14 items. Most of our respondents were white males majoring in computer science and cybersecurity. Since our scale development was inductive, generalizability might be challenging for other IT or STEM majors. Further research is needed to test our scales on other information technology majors. Second, while we established our scale's nomological and predictive validity on other computer programming constructs (i.e., coding grit and coding self-efficacy), we plan to expand its nomological network or path analysis to other established educational motivation and engagement measures. Applying CPEs to another theoretical educational framework will ascertain their usefulness for future educational researchers. Third, while we adhered to the established factor analysis method, our confirmatory factor analysis (CFA) yielded only moderate to good fit indices below .95 for CFA/TFI and above >.05 for RMSEA. Improvement to the instrument will require further data collection. Lastly, learning to program may not occur linearly; in other words, learning strategies may not gradually move up Bloom's levels of knowledge and abilities. Learners could be "overlapping" learning strategies (Dinsmore, 2018), thereby applying all sub-dimensions of CPEs simultaneously.

7. CONCLUSION

In this work, we have developed and validated the CPE scale as a domain-specific strategy. This new scale will be valuable to IT/STEM educators and researchers, who can use it to assess and evaluate student learning and teaching strategies. This work suggests that there are multiple learning strategies for computer programming. Future research may investigate CPE dimensions impact on other existing constructs. Similarly, future research may test CPE constructs on different levels of programming classes (e.g., computer science capstone or software engineering classes) where Visual-CPE is more salient or investigate the associations between CPE and various learning outcomes (performance goals, mastery goals, etc.).

8. REFERENCES

ABET. (2025). *Criteria for Accrediting Computing Programs, 2025 – 2026*. https://www.abet.org/accreditation/accreditation-criteria/criteria-for-accrediting-computing-programs-2025-2026/

Alexander, P. A. (2004). A Model of Domain Learning: Reinterpreting Expertise as a Multidimensional, Multistage Process. In D. Y. Dai & R.J. Sternberg (Eds.), *Motivation, Emotion, and Cognition: Integrative Perspectives on*

Intellectual Functioning and Development (pp. 273-298). Mahwah, NJ: Lawrence Erlbaum Associates.

Asikainen, H., & Gijbels, D. (2017). Do Students Develop Towards More Deep Approaches to Learning During Studies? A Systematic Review on the Development of Students' Deep and Surface Approaches to Learning in Higher Education. *Educational Psychology Review*, 29(2), 205-234. https://doi.org/10.1007/s10648-017-9406-6

Bai, S., Hew, K. F., Sailer, M., & Jia, C. (2021). From Top to Bottom: How Positions on Different Types of Leaderboard May Affect Fully Online Student Learning Performance, Intrinsic Motivation, and Course Engagement. *Computers & Education*, 173, Article 104297.

Bandura, A. (1977). Self-Efficacy: Toward a Unifying Theory of Behavioral Change. *Psychological Review*, 84(2), 191-215. https://doi.org/10.1037/0033-295X.84.2.191

Bennedsen, J., & Caspersen, M. E. (2019). Failure Rates in Introductory Programming: 12 Years Later. *ACM Inroads*, 10(2), 30-36. https://doi.org/10.1145/3324888

Bloom, B. S. (1956). *Taxonomy of Educational Objectives*. NY: Longmans, Green.

Brown, T. A. (2015). *Confirmatory Factor Analysis for Applied Research* (2nd ed.). NYC, NY: Guilford Publications.

Butler, M., Sinclair, J., Morgan, M., & Kalvala, S. (2016). Comparing International Indicators of Student Engagement for Computer Science. *Proceedings of the Australasian Computer Science Week Multiconference* (pp. 1-10). https://doi.org/10.1145/2843043.2843065

Cernosek, G. & Naiburg, E. (2004). *The Value of Modeling* [White paper]. IBM. https://download.boulder.ibm.com/ibmdl/pub/software/dw/library/rational/pdf/valueofmodeling.pdf

Chi, M. T., Feltovich, P. J., & Glaser, R. (1981). Categorization and Representation of Physics Problems by Experts and Novices. *Cognitive Science*, 5(2), 121-152. https://doi.org/10.1207/s15516709cog0502_2

College Board. (n.d.). *SAT Suite of Assessments – Reports*. https://reports.collegeboard.org/sat-suite-program-results

Compeau, D. R., & Higgins, C. A. (1995). Computer Self-Efficacy: Development of a Measure and Initial Test. *MIS Quarterly*, 19(2), 189-211. https://doi.org/10.2307/249688

Compeau, D., Gravill, J., Haggerty, N., & Kelley, H. (2007). Computer Self-Efficacy: A Review. In P. Zhang & D. F. Galletta (Eds.), *Human-Computer Interaction and Management Information Systems: Foundations* (pp. 225-261. New York: Routledge.

Clarke-Midura, J., Sun, C., Pantic, K., Poole, F., & Allan, V. (2019). Using Informed Design in Informal Computer Science Programs to Increase Youths' Interest, Self-Efficacy, and Perceptions of Parental Support. *ACM Transactions on Computing Education*, 19(4), 1-24. https://doi.org/10.1145/3319445

Davies, C. H. (2002). Student Engagement With Simulations: A Case Study. *Computers & Education*, 39(3), 271-282. https://doi.org/10.1016/S0360-1315(02)00046-5

Davis, F. D. (1989). Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology. *MIS Quarterly*, 13(3), 319-340. https://doi.org/10.2307/249008

Dinsmore, D. L. (2018). *Strategic Processing in Education*. New York, NY: Routledge. https://doi.org/10.4324/9781315505732

Duckworth, A. L. (2016). *Grit: The Power of Passion and Perseverance*. NY: Scribner.

Duckworth, A. L., Peterson, C., Matthews, M. D., & Kelly, D. R. (2007). Grit: Perseverance and Passion for Long-Term Goals. *Journal of Personality and Social Psychology*, 92(6), 1087-1101. https://doi.org/10.1037/0022-3514.92.6.1087

Duckworth, A. L., & Quinn, P.D. (2009). Development and Validation of the Short Grit Scale. *Journal of Personality Assessment*, 91(2), 166-174. https://doi.org/10.1080/00223890802634290

Dunlosky, J., Rawson, K. A., Marsh, E. J., Nathan, M. J., & Willingham, D. T. (2013). Improving Students' Learning With Effective Learning Techniques: Promising Directions From Cognitive and Educational Psychology. *Psychological Science in the Public Interest*, 14(1), 4-58. https://doi.org/10.1177/1529100612453266

Fredricks, J. A., Blumenfeld, P. C., & Paris, A. H. (2004). School Engagement: Potential of the Concept, State of Evidence. *Review of Educational Research*, 74(1), 59-109. https://doi.org/10.3102/00346543074001059

Fredricks, J. A., & McColskey, W. (2012). The Measurement of Student Engagement: A Comparative Analysis of Various Methods and Student Self-report Instruments. In S. L. Christenson, A. L. Reschly, & C. Wylie (Eds.), *Handbook of Research on Student Engagement* (pp. 763-782). Springer US. https://doi.org/10.1007/978-1-4614-2018-7_37

Giannakos, M. N., Pappas, I. O., Jaccheri, L., & Sampson, D. G. (2017). Understanding Student Retention in Computer Science Education: The Role of Environment, Gains, Barriers and Usefulness. *Education and Information Technologies*, 22(5), 2365-2382. https://doi.org/10.1007/s10639-016-9538-1

Greene, B. A. (2015). Measuring Cognitive Engagement With Self-Report Scales: Reflections From Over 20 Years of Research. *Educational Psychologist*, 50(1), 14-30. https://doi.org/10.1080/00461520.2014.989230

Gunness, A., Matanda, M. J., & Rajaguru, R. (2023). Effect of Student Responsiveness to Instructional Innovation on Student Engagement in Semi-Synchronous Online Learning Environments: The Mediating Role of Personal Technological Innovativeness and Perceived Usefulness. *Computers & Education*, 205, Article 104884. https://doi.org/10.1016/j.compedu.2023.104884

Hair, J. F., Anderson, R. E., Tatham, R. L., & Black, W. C. (1995). *Multivariate Data Analysis with Readings* (4th ed.). Upper Saddle River, NJ: Prentice-Hall.

Hagger, M. S., Gucciardi, D. F., & Chatzisarantis, N. L. D. (2017). On Nomological Validity and Auxiliary Assumptions: The Importance of Simultaneously Testing Effects in Social Cognitive Theories Applied to Health Behavior and Some Guidelines. *Frontier in Psychology*, 8, Article 1933. https://doi.org/10.3389/fpsyg.2017.01933

Harrington, D. (2009). *Confirmatory Factor Analysis*. New York, NY: Oxford University Press. https://doi.org/10.1093/acprof:oso/9780195339888.001.0001

Hazzam, J., & Wilkins, S. (2023). The Influences of Lecturer Charismatic Leadership and Technology Use on Student Online Engagement, Learning Performance, and Satisfaction. *Computers & Education*, 200, Article 104809. https://doi.org/10.1016/j.compedu.2023.104809

Hsiao, J.-C., Chen, S.-K., Chen, W., & Lin, S. S. (2022). Developing a Plugged-in Class Observation Protocol in High-school Blended STEM Classes: Student Engagement, Teacher Behaviors and Student-Teacher Interaction Patterns. *Computers & Education*, 178, Article 104403. https://doi.org/10.1016/j.compedu.2021.104403

Igbaria, M. & Baroudi, J. J. (1993). A Short-Form Measure of Career Orientations: A Psychometric Evaluation. *Journal of Management Information Systems*, 10(2), 131-154. https://doi.org/10.1080/07421222.1993.11518003

Kanaparan, G., Cullen, R., & Mason, D. (2013). Self-Efficacy and Engagement as Predictors of Student Programming Performance. *PACIS 2013 Proceedings*, Article 282. https://aisel.aisnet.org/pacis2013/282

Kahu, E. R. (2013). Framing Student Engagement in Higher Education. *Studies in Higher Education*, 38(5), 758-773. https://doi.org/10.1080/03075079.2011.598505

Kornell, N. (2009). Optimising Learning Using Flashcards: Spacing is More Effective Than Cramming. *Applied Cognitive Psychology*, 23(9), 1297-1317. https://doi.org/10.1002/acp.1537

Kuh, G. D. (2001). Assessing What Really Matters to Student Learning Inside the National Survey of Student Engagement. *Change: The Magazine of Higher Learning*, 33(3), 10-17. https://doi.org/10.1080/00091380109601795

Lai, H.-M. (2021). Understanding What Determines University Students' Behavioral Engagement in a Group-based Flipped Learning Context. *Computers & Education*, 173, Article 104290. https://doi.org/10.1016/j.compedu.2021.104290

Leidig, P. M., & Anderson, G. (2020). Updating the Information Systems Curriculum: The ACM/AIS IS2020 Joint Project. *Proceedings of the 2020 Computers and People Research Conference* (pp. 8-9). https://doi.org/10.1145/3378539.3393870

Li, S., & Lajoie, S. P. (2022). Cognitive Engagement in Self-Regulated Learning: An Integrative Model. *European Journal of Psychology of Education*, 37(3), 833-852. https://doi.org/10.1007/s10212-021-00565-x

Lorås, M., Sindre, G., Trætteberg, H., & Aalberg, T. (2022). Study Behavior in Computing Education—A Systematic Literature Review. *ACM Transactions on Computing Education*, 22(1), 1-40. https://doi.org/10.1145/3469129

Mahatanankoon, P. (2018). Exploring the Antecedents to Computer Programming Self-Efficacy. *Proceedings of the 10th International Conference on Advances in Information Technology* (pp. 1-6). https://doi.org/10.1145/3291280.3291791

Mahatanankoon, P., & Sikolia, D. W. (2017). Intention to Remain in a Computing Program: Exploring the Role of Passion and Grit. AMCIS 2017 Proceedings, Article 17. https://aisel.aisnet.org/amcis2017/ISEducation/Presentations/17

Mahatanankoon, P., & Wolf, J. (2021). Cognitive Learning Strategies in an Introductory Computer Programming Course. *Information Systems Education Journal*, 19(3), 11-20.

Mathieson, K. (1991). Predicting User Intentions: Comparing the Technology Acceptance Model With the Theory of Planned Behavior, *Information Systems Research*, 2(3), 173-191. https://doi.org/10.1287/isre.2.3.173

Mayer, R. E. (2002). Rote Versus Meaningful Learning. *Theory Into Practice*, 41(4), 226-232. https://doi.org/10.1207/s15430421tip4104_4

McClendon, C., Neugebauer, R. M., & King, A. (2017). Grit, Growth Mindset, and Deliberate Practice in Online Learning. *Journal of Instructional Research*, 6, 8-17. https://doi.org/10.9743/JIR.2017.2

Menon, P. (2023). Teaching Tip: An Example-based Instructional Method to Develop Students' Problem-Solving Efficacy in an Introductory Programming Course. *Journal of Information Systems Education*, 34(1), 1-15.

Miller, R. B., Greene, B. A., Montalvo, G. P., Ravindran, B., & Nichols, J. D. (1996). Engagement in Academic Work: The Role of Learning Goals, Future Consequences, Pleasing Others and Perceived Ability. *Contemporary Educational Psychology*, 21(4), 388-422. https://doi.org/10.1006/ceps.1996.0028

Morgan, M., Butler, M., Thota, N., & Sinclair, J. (2018a). How CS Academics View Student Engagement. *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education* (pp. 284-289). https://doi.org/10.1145/3197091.3197092

Morgan, M., Sinclair, J., Butler, M., Thota, N., Fraser, J., Cross, G., & Jackova, J. (2018b). Understanding International Benchmarks on Student Engagement: Awareness and Research Alignment from a Computer Science Perspective. *Proceedings of the 2017 ITiCSE Conference on Working Group Reports* (pp. 1-24). https://doi.org/10.1145/3174781.3174782

Newman, F. M., Wehlage, G. G., Lamborn, S. D. (1992). The Significance and Sources of Student Engagement. In: F. M. Newman (Ed.), *Student Engagement and Achievement in American Secondary Schools* (pp. 11-39). New York: Teachers College Press.

Novak, J. D., & Canas, A. J. (2008). *The Theory Underlying Concept Maps and How to Construct and Use Them*. Technical Report. Pensacola, FL: Institute of Human and Machine Cognition.

Nunnally, J. C. (1978). *Psychometric Theory* (2nd ed.). New York: McGraw-Hill.

Obaido, G., Agbo, F. J., Alvarado, C., & Oyelere, S. S. (2023). Analysis of Attrition Studies Within the Computer Sciences. *IEEE Access*, 11. https://doi.org/10.1109/ACCESS.2023.3280075

Pellas, N., Zhang, H., Lin, L., Zhan, Y., & Ren, Y. (2024). Assessing Computational Thinking, Motivation, and Grit of Undergraduate Students Using Educational Robots, The Impact of Teaching Presence on Online Engagement Behaviors. *Journal of Educational Computing Research*, 62(2), 620-644. https://doi.org/10.1177/07356331231210946

Pett, M. A., Lackey, N. R., & Sullivan, J. J. (2003). *Making Sense of Factor Analysis*. Thousands Oak, CA: Sage Publications, Inc. https://doi.org/10.4135/9781412984898

Rawson, K. A., & Kintsch, W. (2005). Rereading Effects Depend on Time of Test. *Journal of Educational Psychology*, 97(1), 70-80. https://doi.org/10.1037/0022-0663.97.1.70

Renumol, V. G., Janakiram, D., & Jayaprakash, S. (2010). Identification of Cognitive Processes of Effective and Ineffective Students During Computer Programming. *ACM Transactions on Computing Education*, 10(3), 1-21. https://doi.org/10.1145/1821996.1821998

Saqr, M., López-Pernas, S., Helske, S., & Hrastinski, S. (2023). The Longitudinal Association Between Engagement and Achievement Varies by Time, Students' Profiles, and Achievement State: A Full Program Study. *Computers & Education*, 199, Article 104787. https://doi.org/10.1016/j.compedu.2023.104787

Schwarz, C. & Zhu, Z. (2015). The Impact of Student Expectations in Using Instructional Tools on Student Engagement: A Look Through the Expectation Disconfirmation Theory Lens. *Journal of Information Systems Education*, 26(1), 47-58.

Silva, L., Mendes, A., Gomes, A., & Fortes, G. (2024). What Learning Strategies Are Used by Programming Students? A Qualitative Study Grounded on the Self-Regulation of Learning Theory. *ACM Transactions on Computing Education*, 24(1), 1-26. https://doi.org/10.1145/3635720

Sinclair, J., Butler, M., Morgan, M., & Kalvala, S. (2015). Measures of Student Engagement in Computer Science. *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education* (pp. 242-247). https://doi.org/10.1145/2729094.2742586

Straub, D., Boudreau, M-C., & Gefen, D. (2004). Validation Guidelines for IS Positivist Research. C*ommunications of the AIS*, 13(1), Article 24, 380-427. https://doi.org/10.17705/1CAIS.01324

Tsai, C.-Y., Chen, Y.-A., Hsieh, F.-P., Chuang, M.-H., & Lin, C.-L. (2023). Effects of a Programming Course Using the GAME Model on Undergraduates' Self-Efficacy and Basic Programming Concepts. *Journal of Educational Computing Research*, 62(3), 1-23. https://doi.org/10.1177/07356331231206071

Vrugt, A., & Oort, F. J. (2008). Metacognition, Achievement Goals, Study Strategies and Academic Achievement: Pathways to Achievement. *Metacognition and Learning*, 3(2), 123-146. https://doi.org/10.1007/s11409-008-9022-4

Westrick, P. A., Marini, J. P., & Shaw, E. J. (2021). Using SAT® Scores to Inform Academic Major-Related Decisions and Planning on Campus. *College Board*. https://eric.ed.gov/?id=ED613434

Wolf, J. R., & Jia, R. (2015). The Role of Grit in Predicting Student Performance in Introductory Programming Courses: An Exploratory Study. *SAIS 2015 Proceedings*, Article 21. https://aisel.aisnet.org/sais2015/21/

Wolf-Wendel, L., Ward, K., & Kinzie, J. (2009). A Tangled Web of Terms: The Overlap and Unique Contribution of Involvement, Engagement, and Integration to Understanding College Student Success. *Journal of College Student Development*, 50(4), 407-428. https://doi.org/10.1353/csd.0.0077

Wong, Z. Y., & Liem, G. A. D. (2022). Student Engagement: Current State of the Construct, Conceptual Refinement, and Future Research Directions. *Educational Psychology Review*, 34(1), 107-138. https://doi.org/10.1007/s10648-021-09628-3

Yu, Z., Gao, M., & Wang, L. (2021). The Effect of Educational Games on Learning Outcomes, Student Motivation, Engagement and Satisfaction. *Journal of Educational Computing Research*, 59(3), 522-546. https://doi.org/10.1177/0735633120969214

Zhang, H., Lin, L., Zhan, Y., & Ren, Y. (2016). The Impact of Teaching Presence on Online Engagement Behaviors. *Journal of Educational Computing Research*, 54(7), 887-900. https://doi.org/10.1177/0735633116648171

Zhang, X., Crabtree, J. D., Terwilliger, M. G., & Jenkins, J. T. (2020). Teaching Tip: Teaching Introductory Programming from A to Z: Twenty-Six Tips From the Trenches. *Journal of Information Systems Education*, 31(2), 106-118.

Zhong, L. (2023). Investigating Students' Engagement Patterns and Supporting Game Features in a Personalized Computerized Role-Playing Game Environment. *Journal of Educational Computing Research*, 61(3), 578-604. https://doi.org/10.1177/07356331221125946

AUTHOR BIOGRAPHIES

Pruthikrai Mahatanankoon is a professor of information systems in the School of Information Technology at Illinois State University. He teaches various information systems courses, including systems analysis and design. His research interests include IT education, mobile commerce, and workplace IT adoption. His research often integrates psychological and educational theories with information systems and has been published in various academic journals and conferences.

James R. Wolf is a professor of information systems in the School of Information Technology at Illinois State University. His research interests include ethical AI, AI in human decision making, technology-enhanced education, and student engagement. His broader scholarly interests include AI applications in healthcare, neural diversity in IT, and blockchain governance frameworks.

APPENDICES

Appendix A. Original Cognitive Programming Engagement (CPE) Scale

Please read the following statements, and for each, select the answer that best represents your learning strategies in your most recent programming class. *1=not like me at all, 2=not much like me, 3=somewhat like me, 4=mostly like me, 5=very much like me*

Shallow Learning Strategies
SL1: I try to memorize the steps for solving programming problems presented in the text or in the lecture. *(dropped; CFA)*
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. (*dropped*)

Deep Learning Strategies
DL1: When studying, I try to combine different pieces of information from course material in new ways. (*dropped; PCA*)
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 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. (*dropped; PCA*)
DL6: I classify programming problems into categories before I begin to work them. (*dropped; PCA*)
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 (*dropped; PCA*)
DL12: Some "programming" problems can be visualized using diagrams or models.
DL13: I develop models or pictures to help me visualize how programming work.
DL14: I model different program modules or functions using some diagramming techniques.
DL15: I use some diagramming techniques to understand how programming work (*dropped; CFA).*
DL16: When I finish my programming practice questions/problems I check my solution for semantic errors. (*dropped; CFA*)

Appendix B. Final Cognitive Programming Engagement (CPE) Dimensions

Cognitive Programming Engagement (CPE)

Shallow-CPE *(cpe-S, α =.821)*

- When I study for the tests, I review my class notes and look at solved programming problems.
- 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.
- I find reviewing previously solved programming problems to be a good way to study for a test.

Practical-CPE *(cpe-P, α =.809)*

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

Analytical-CPE *(cpe-A, α =.782)*

- When I work on a programming problem, I analyze it to see if there is more than one way to get the right solution.
- While learning new programming concepts, I try to think of practical applications.
- I put together programming ideas or concepts and draw conclusions that were not directly stated in course materials.

Visual-CPE *(cpe-V, α =.881)*

- I draw pictures or diagrams to help me solve some programming problems.
- Some "programming" problems can be visualized using diagrams or models.
- I develop models or pictures to help me visualize how programming works.
- I model different program modules or functions using some diagramming techniques.
- I use some diagramming techniques to understand how programming works.

Nomological Validity Constructs

Coding Self-Efficacy (*CSE, α =.869*)

- Compared to others in my programming class, I am confident in my programming skills.
- Compared to others in my programming class, I am confident of my technical skills.
- I have faith in my ability to learn new programming skills constantly.
- Compared to others in my programming class, I am confident of my programming techniques.

Coding Grit (*CGR*, see Mahatanankoon & Sikolia, 2017, adapted from Duckworth et al., 2007; Duckworth & Quinn, 2009)

Appendix C. Correlation Matrix

	AV	SD	SL1	SL2	SL3	SL4	SL5	DL1	DL2	DL3	DL4	DL5	DL6	DL7	DL8	DL9	DL 10	DL 11	DL 12	DL 13	DL 14	DL 15	DL 16
SL1	3.04	1.05	1.00	**.434**	**.551**	**.376**	**.238**	**.304**	.099	**.152**	.116	-.004	.109	.098	-.022	**-.187**	-.059	**.219**	**.162**	.115	**.162**	**.220**	.123
SL2	3.70	1.08		1.00	**.621**	**.641**	**.206**	**.367**	.135	**.224**	**.245**	**.173**	**.183**	**.171**	.150	.032	**.151**	.121	**.243**	.068	.034	.102	**.217**
SL3	3.31	1.22			1.00	**.561**	**.260**	**.204**	.079	.125	.078	.063	**.208**	.150	-.018	-.150	.038	**.174**	.090	-.03	.072	**.160**	**.192**
SL4	3.67	1.10				1.00	**.282**	**.195**	-.074	-.014	-.005	**.289**	**.152**	.014	-.022	**-.188**	.115	**.208**	.053	.038	.025	.088	.083
SL5	2.41	1.01					1.00	.120	.093	**.150**	.096	.112	**.191**	-.042	**-.202**	**-.189**	.091	.026	-.018	.047	.077	.136	.021
DL1	3.21	1.03						1.00	**.251**	**.480**	**.414**	**.187**	**.282**	**.242**	**.337**	**.310**	**.331**	.102	**.284**	.120	**.281**	**.252**	**.269**
DL2	2.69	1.26							1.00	**.319**	**.254**	.080	**.211**	.012	**.211**	**.157**	**.233**	-.057	**.532**	**.631**	**.599**	**.615**	**.159**
DL3	3.00	1.11								1.00	**.619**	**.259**	**.325**	**.305**	**.220**	**.197**	**.541**	**.176**	**.180**	**.299**	**.381**	**.271**	**.283**
DL4	3.23	1.09									1.00	**.256**	**.261**	**.285**	**.331**	**.191**	**.599**	.125	**.224**	.119	**.210**	**.183**	**.236**
DL5	3.89	0.99										1.00	.130	**.260**	**.230**	**.157**	**.240**	.038	.067	.099	.106	.149	.082
DL6	2.55	1.05											1.00	**.226**	**.231**	**.214**	**.248**	.147	.076	**.244**	**.340**	**.320**	**.247**
DL7	3.38	1.07												1.00	**.526**	**.463**	**.225**	**.326**	**.223**	.029	**.159**	.141	**.426**
DL8	3.78	1.03													1.00	**.648**	**.228**	**.173**	**.347**	.067	**.217**	**.250**	**.320**
DL9	3.26	1.04														1.00	**.259**	.082	**.296**	.055	**.180**	**.226**	**.299**
DL10	3.21	1.04															1.00	.146	**.225**	**.225**	**.318**	**.233**	**.163**
DL11	3.80	1.21																1.00	**.270**	.115	.106	.087	**.638**
DL12	3.33	1.06																	1.00	**.527**	**.512**	**.534**	**.385**
DL13	2.73	1.07																		1.00	**.739**	**.654**	**.234**
DL14	2.50	0.92																			1.00	**.759**	**.272**
DL15	2.63	0.98																				1.00	**.283**
DL16	3.78	1.05																					1.00

Bolded correlation values signify significant relationships of at least .05, one-tailed.

Appendix D. Factor Loadings

CPE Items	CPE Dimensions					
	Factor 1: Shallow	**Factor 2: Visual**	**Factor 3: Practical**	**Factor 4: Analytical**	**Factor 5: Compiling**	**α (if deleted)**
SL1	.687					.821
SL2	.843					.748
SL3	.851					.739
SL4	.806					.776
DL2		.805				.864
DL12		.673				.880
DL13		.867				.844
DL14		.853				.844
DL15		.847				.846
DL3			.810			.749
DL4			.845			.701
DL10			.803			.765
DL7				.651		.787
DL8				.855		.633
DL9				.850		.689
DL11					.910	NA
DL16					.785	NA
Reliability α	.820	.881	.809	.782	.774	

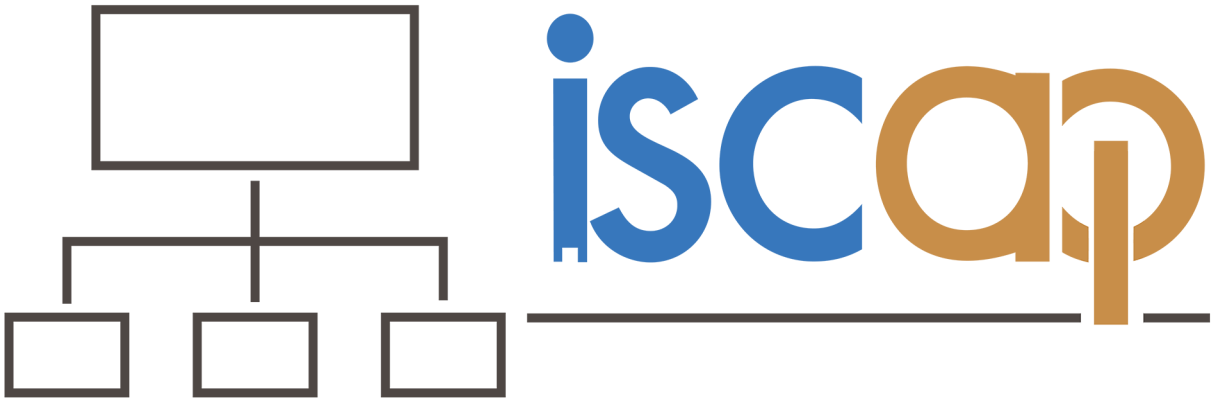
*Factor loadings more than 0.4 are shown. NA = Fewer than three items.

Appendix E. Intercorrelations: Factor-Based Scales of Computer Programming Engagement (CPE) (N=120)

Dimension	Mean	S.D.	Shallow	Practical	Analytical	Visual	AVE	Composite Reliability
Shallow (SL2-SL4)	3.56	.973	1	.144	.021	.100	.610	.821
Practical (DL3, DL4, DL10)	3.17	.918		1	.351**	.347**	.590	.809
Analytical (DL7-DL9)	3.47	.874			1	.243**	.561	.782
Visual (DL2, DL12-DL15)	2.78	.875				1	.617	.881

**p < .001

INFORMATION SYSTEMS & COMPUTING ACADEMIC PROFESSIONALS



STATEMENT OF PEER REVIEW INTEGRITY

All papers published in the ***Journal of Information Systems Education*** have undergone rigorous peer review. This includes an initial editor screening and double-blind refereeing by three or more expert referees.

ISSN: 2574-3872 (Online) 1055-3096 (Print)