Navigating Strategies in Enacting Evidence-Based Teaching of Introductory Computing for Engineers

Michael L. Falk
Professor of Materials Science and Engineering, Mechanical Engineering, and Physics
Vice Dean for Undergraduate Education, Whiting School of Engineering
Johns Hopkins University, Baltimore, MD USA

Alejandra J. Magana
Associate Professor of Computer and Information Technology, and Engineering Education
Purdue University, West Lafayette, IN USA

Michael J. Reese, Jr.
Associate Dean of University Libraries
Director, Center for Educational Resources
Johns Hopkins University, Baltimore, MD USA

Abstract

A particular institutional change to the teaching of computing within a school of engineering is discussed in light of the scholarship on theories of change within educational institutions. Particular attention is paid to how these various change strategies may be deployed in succession or concurrently to bring educational innovations from the classroom to the institutional scale. The role of the initiatives of individual scholars, communities of practitioners, and institutional leaders as change agents are seen to interact in ways that evolve and shift throughout the course of this particular institutional change.

Contents

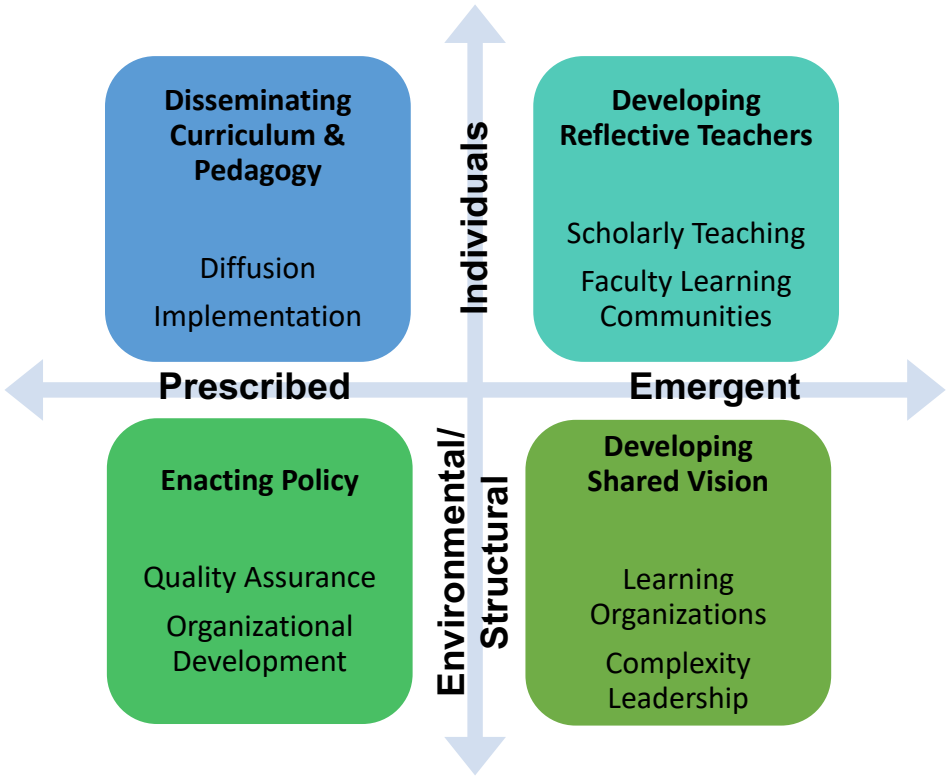

Figure 1 Categories of Change Strategies. Adapted from Borrego & Henderson, 2014.

I. Introduction

As the field of engineering education has matured, a conversation has emerged as to how to categorize and analyze mechanisms of change within higher education that have led to innovations in engineering teaching practices and learning outcomes. The advantage of having clearly articulated theories of change is that such theories can clarify distinctions between strategies, thereby serving to guide effective innovation initiatives. Furthermore, such theories provide a means of utilizing information gathered about the pre-conditions, implementation, and assessment regarding an intervention, to infer the critical elements for a particular strategy's success.

Here we consider a particular instance of an on-going institutional change in the engineering education realm using the framework of Borrego and Henderson (Borrego & Henderson, 2014). This framework characterizes eight possible strategies for instructional change. These eight strategies can be classified under four master categories defined by two axes as shown in Figure 1, which were articulated in prior work (Henderson, 2010; 2011). Two strategies involve the prescriptive promotion of well-tested educational methodologies by individuals via curriculum and pedagogical shifts. Within this category, *diffusion strategies* focus on the sharing of information, while *implementation strategies* establish more structured support for such adoption efforts. Two strategies are identified that promote development of new educational approaches or adaptation of existing approaches in novel ways by individuals, termed reflective teacher strategies. S*cholarly teaching* approaches target efforts by one or a small group of faculty, while *faculty learning communities* engage multiple faculty through a supportive relationship to foster growth. Attempts to promote educational innovation by structural or environmental changes across institutions are distinguished from those targeting one or a small

number of instructors. These are also divided into those focusing on prescribed efforts, so called policy strategies, and shared vision strategies that focus on innovation. *Quality assurance* is one such policy strategy in which assessment are undertaken and compliance rewarded. The *organizational development* approach to policy strategy, by contrast, relies on a leader or leadership team to craft a new vision to be attained by planned changes to the organizational structure and/or practices. Finally, two shared vision strategies that engage at the organizational level to encourage innovation are discussed. *Learning organizations* are envisioned as the creation of both formal and informal groups within the organizational structure whose goal is to promote innovation. A *complexity leadership theory* approach, only briefly discussed in this prior work, encourages innovative disruption at all scales with an eye toward encouraging the best ideas that emerge and allowing these to converge into new organizational practices. It is our goal in this work to discuss, within the context of one particular on-going institutional change, how multiple strategies were deployed and how their enactments served to propel the change process. As discussed by Borrego and Henderson, systemic change is rarely achieved by deploying only one such strategy. We hope that this case study will provide a useful reference point for thinking about how instructional change in engineering education can translate from individual classrooms to a larger organization.

The goal of each phase of the instructional innovation described in this case study is enacting evidence-based teaching methods in the context of teaching introductory computing to engineers. Preparation of engineers across all disciplines now requires that students develop a strong facility to deploy computing as a tool, as a means of tool development, and as a pathway to discovery and understanding. As such, the learning objectives associated with introductory computing now often include, but are not limited to:

- conveying a fundamental understanding of computing and programming constructs,
- providing experiences that foster capacity for algorithmic thinking and design,
- developing a facility to perform particular disciplinarily relevant computing tasks,
- increasing analytical ability to support problem solving and design using computing,
- leveraging computing knowledge to better understand disciplinary concepts, and
- motivating students to incorporate computing into their education and future careers.

In order to attain these learning objectives, various strategies have been deployed to introduce engineering students to computing and help them gain mastery both through Computer Science coursework and through disciplinarily-embedded coursework that focus on the utility of computing in a specific engineering context.

During the past seven years Falk has played a change-agent role in a collective rethinking of the way computing is integrated into undergraduate engineering education at Johns Hopkins University (JHU) in Computer Science courses and other engineering disciplines. This process started with the development of a new class, which resulted in a curricular change affecting multiple classes within a single engineering major. The process culminated in a revision of the organizational structure, teaching methodology and curriculum across 7 of the 10 departments in the Whiting School of Engineering. This change is on-target to eventually engage the entire engineering school and, potentially, the majority of undergraduates across the institution. The change strategies deployed over the past seven years could be characterized in its early stages as a “Scholarly Teaching” strategy aimed at developing a reflective teacher, but later evolved into an “Organizational Development” strategy aimed at enacting a change in policy that, in turn, has evolved into a “Faculty Learning Community” change strategy. It could also be argued that the entire innovation arose from a “Complexity Leadership” strategy that is embedded in the

school's organizational structure and which facilitated the development of a shared vision. In the context of this particular set of experiences, we ask questions about how these various strategies inform each other. We further consider the ways in which shifts between strategies were accomplished in order to advance systemic change.

II. Background and Rationale

Computational thinking skills and practices, although primarily conceived by the Computer Science community, have also been embraced by engineering educators in elementary and higher education. We *operationalize* computational thinking as a set of practices including (Weintrop et al., 2014): (a) data and information skills including generating, collecting, manipulating, analyzing, and visualizing data; (b) modeling and simulation skills associated with activities like understanding, using, assessing, testing, and building or extending computational models, (c) problem solving skills comprising activities such as troubleshooting and debugging computing artifacts, assessing solutions to problems, applying problem solving strategies, and creating abstractions; and (d) systems thinking skills including investigating and visualizing systems, identifying relationships and levels of systems, and managing complexity. We note that of these (i.e., a-d), only particular aspects of (a) and (c), such as manipulating data, abstracting problems and creating computing artifacts, are routinely and extensively developed in traditional introductory computing courses within Computer Science. Introductory computing in engineering, in contrast, tends to focus on different aspects of (a) and (b), particularly data analysis and visualization, as well as modeling and simulation (Magana & Coutinho, 2017).

Computational-thinking practices are well-aligned with current curricular outcomes desired in engineering education. For instance, the Transforming Undergraduate Education in Engineering report ([ASEE], 2013) indicated that industry professionals highly value programming skills and the ability to use computational tools to support problem solving. Similarly, ABET (2013) indicated that the ability to use the techniques, skills, and modern engineering tools is a necessary student outcome for engineering practice. Also, The International Engineering Alliance (2011) indicated the ability to create, select and apply appropriate techniques, resources, and modern engineering tools, including prediction and modeling, to complex engineering activities, with an understanding of the limitations, as highly desirable skills. It is then imperative that the engineering education community identifies effective ways to integrate computation sooner, better and with greater success.

III. Getting the Ball Rolling with a "Scholarly Teaching" Strategy

The initial stage of the innovation we detail here began in the mode of a "Scholarly Teaching" strategy. By 2010 Falk had eight years of experience teaching introductory computing both to large lecture classes of 180-225 students at University of Michigan and then, since moving to Johns Hopkins in 2008, had taught a more in-depth class for upper-division undergraduates and novice graduate students. By 2010 the class in question included a number of innovations, some which had been derived from the prior experiences of the instructor and others which had been motivated by education research that the instructor had become aware of through a number of professional development opportunities. These had included offerings of the Center for Research in Learning and Teaching at University of Michigan, under the aegis of Dr. Cynthia Finelli, through Johns Hopkins' Center for Educational Research, under the leadership of Dr. Candice Dalrymple, and at the Frontiers of Engineering Education (FOEE) workshop of the National Academies. These innovations included the use of disciplinarily-grounded examples as

a way of motivating students' programming efforts and an active-learning methodology for teaching programming concepts enabled by a flipped-classroom structure.

In 2010, after returning from an FOEE workshop, Falk reached out to Dr. Alejandra Magana at Purdue University as a potential partner for engaging in education research to study students' learning processes and outcomes in the context of a new class he was developing called Computation and Programming for Materials Scientists and Engineers (CPMSE). This class was envisioned as the first part of a cross-curricular innovation that would integrate computation in the core coursework for all Materials Science and Engineering majors. The initial course was meant to be an introduction to programming taught in MATLAB® and contextualized with simulation projects germane to materials science. This would lay the groundwork for the further introduction of computational modules, one- or two-week long learning modules in which students utilized computation to acquire a disciplinary core concept and reinforce their computational knowledge. These modules would be woven into the six core lecture courses that every student was required to pass. Funding for this work, focusing on *curriculum development, implementation and assessment*, was provided by the National Science Foundation through a Research Initiation Grant (EEC#1137006), which then evolved to a focus on *pedagogy* through a Research in Engineering Education Grant (EEC#1329262).

Magana's primarily role was to oversee the design of the educational research projects. She suggested relevant education theory to guide the research questions, developed and suggested data collection instruments, and analyzed the data with a team of graduate students. She worked closely with Falk, who provided the team with relevant context for his engineering course and ultimately decided which strategies to integrate into his course. Michael Reese, an instructional designer at the JHU Center for Educational Resources, also assisted in research design including developing data collection instruments. He also oversaw local data collection to ensure student anonymity. The group collaboratively reviewed the results used to inform changes to future implementations of the course.

Logistical changes along with administrative support were also required to undertake this work. For instance, Falk was provided with a semester of teaching release to design and implement learning modules in the six core courses in collaboration with the instructors for each course. Also, after the initial implementation a special teaching assistant with computational expertise was assigned to be responsible for supporting the continued implementation of these modules in future semesters. Discussion of the computational modules and their alignment with the course content was incorporated into annual discussions of course outcomes routinely undertaken as part of the department's iterative improvement process, necessary for compliance with ABET accreditation standards. Eventually Falk was able to share responsibility for teaching the CPMSE course with other instructors who also had strong computationally-focused academic backgrounds.

A facilitating component of this entire process has been the use of an evidence-based approach for continuous improvement. That is, as we transitioned throughout this process we systematically conducted design-based research in the classroom to inform curricular and pedagogical decisions. Design-based research is an appropriate methodology because it focuses on understanding learning in real-world practice (Barab & Squire, 2004). Another characteristic of design-based research that makes it appropriate for this project is that it is contextual (Wang & Hannafin, 2005). It considers education as an applied field where researchers have transformative agendas (Barab & Squire, 2004). As such, they develop contexts, frameworks, tools, and pedagogical models with the intent to produce new theories, artifacts, and practices

that can impact teaching, learning, and engagement in naturalistic settings (Barab & Squire, 2004). Design-based research provides us with a series of approaches that will allow us to "engineer" and study particular forms of learning that will be subject to test, revision, and iteration among the products of this research (Cobb et al., 2003).

This preliminary work identified the effect upon student learning and student attitudes of the CPMSE course. This course taught programming in an active-learning environment wherein students applied computing skills by creating simulations of disciplinarily relevant processes and systems. This study indicated that students achieved high levels of motivation for computing and an enhanced sense of their capacity to deploy computing (Magana, Falk, & Reese, 2013). A related study determined that these students also showed a higher capacity to use computing to acquire disciplinary knowledge through programming exercises in later coursework (Magana, Falk, Vieira, & Reese, 2016). Additional work examined the role of additional scaffolded supports in permitting students of various backgrounds to succeed in the challenging task of integrating newly acquired disciplinary expertise with newly acquired programming skills (Vieira, Magana, Roy, Falk, & Reese, 2016a). This study indicated that skills integrating disciplinary knowledge with computing practice could be fostered by intentional introduction of particular course practices (Vieira, Magana, Falk, & Garcia, 2017). These findings strongly suggest that students' early exposure to a contextualized and active approach to introductory computing has distinct advantages for student learning and subsequent curricular integration of computing. However, concomitant challenges were also identified.

Once the curricular innovation was created, the next step in the process was to focus on pedagogy in order to mitigate some of the challenges of the approach. The main issue identified was that a significant number of students experienced "cognitive overload," a concept we will discuss more below, due to the complexity of integrating new computing skills with unfamiliar and equally novel disciplinary concepts. In particular, the mapping of physical systems to mathematical constructs and the translation of these mathematical models into programs was a significant hurdle for students. Multiple pedagogical and learning strategies were implemented to guide and support students through this complex learning process. To summarize and present this work, we introduce the cognitive apprenticeship model (Collins, Brown, & Newman, 1989) as an organizational framework that helps to locate this work within the existing literature. The cognitive apprenticeship model (see Figure 2), is an instructional approach that combines the cognitive and metacognitive skills required for developing expertise. This model proposes the design of learning environments that merge "the content being taught, the pedagogical methods employed, the sequencing of learning activities, and the sociology of learning" (Collins et al., 1989, p. 3).

Content refers to the types of knowledge required for expertise. *Method* refers to the pedagogical approach, learning strategies, or teaching methods used. *Sequencing* refers to designing the structure and the order of the tasks so as to optimize meaningful student engagement. *Sociology* refers to the context within which learning experiences are situated via the application of skills to realistic problems. The following sub-sections of the cognitive apprenticeship model present a sample of effective practices for integrating the teaching of computer programming within the context of disciplinary computation.

III.A Content

Traditionally courses designed to teach computational thinking practices, typically within departments of Computer Science, have a strong focus on programming concepts, principles and

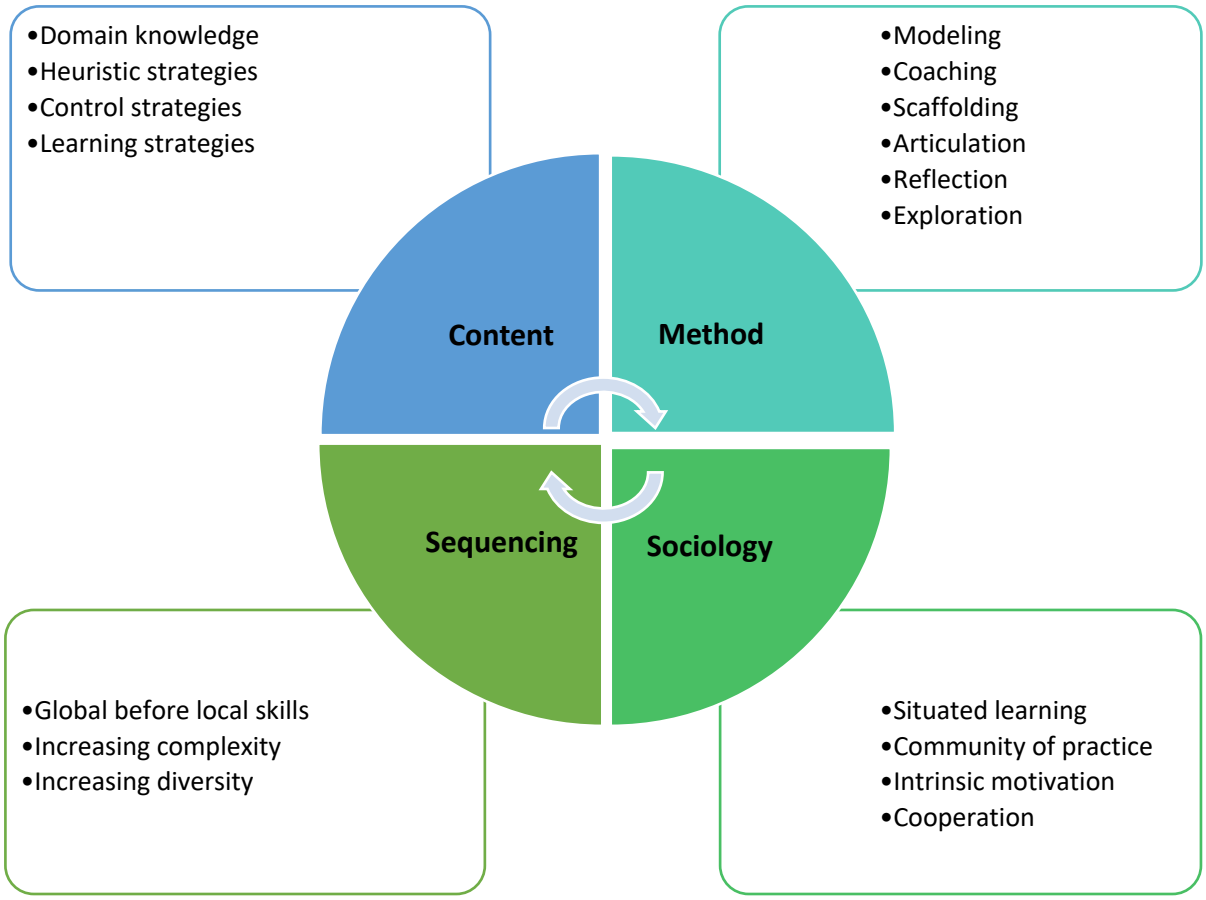

Figure 2 Elements of the cognitive apprenticeship model

procedures (Tew & Guzdial, 2010). Many introductory computing courses that have been newly developed within engineering disciplines continue to teach much of this basic programming knowledge, but expand the scope of their courses to include additional topics (e.g., Narayanan, 2007; Stickel, 2011). In many courses these include applications of computing in specific contexts such as data visualization, data analysis techniques such as linear regression, solutions of linear equations, and methods of solution of differential equations and partial differential equations (Morris, Long, Haghighat, & Brady, 1996). In other courses the focus expands further to include linking mathematical modeling practices to algorithmic representations in the form of simulations (Magana et al., 2013). A few even touch on tool building in the form of the development of graphical user interfaces.

While expanding the range of computational learning objectives is an imperative for engineering disciplines that wish to incorporate computation into subsequent coursework, this expansion runs significant risks. The most obvious risk is the limitation posed by time constraints. Introducing additional disciplinarily-focused learning objectives may cause the abandonment of others that are critical to the development of computational thinking. Furthermore, literature in computer science education has identified for a long time that learning to program is difficult (Lister et al., 2004; McCracken et al., 2001; Soloway & Spohrer, 1989). Some of the difficulties learners experience include identifying: (i) the purpose of the programming task; (ii) the general properties or functionality of the machine that one is intending

to control; (iii) the syntax and semantics of the programming language; (iv) structure, where the learner needs to deal with the difficulties of acquiring standard patterns or schemas that can be implemented to attain small-scale goals; and (v) pragmatics, where the learner develops the skills to be able to specify, develop, test and debug programs using whatever tools are available (Du Boulay, 1986; Pea & Kurland, 1983).

Our prior research also indicates that in addition to programming challenges, certain practices, particularly those that demand the integration of differing representations, i.e. physical, mathematical and algorithmic such as occur in modeling and simulation, run significant risks of causing "cognitive overload" (Magana, Falk & Reese, 2013; Vieira, Roy, Magana, Falk & Reese, 2016). In such cases student short-term memory is insufficient to successfully undertake complex processes that inter-relate multiple novel concepts.

III.B Method

The dimension of "method" is particularly broad in the context of computing education. At the time this innovation was initiated, practices at JHU included classes that ranged in size from 20 students to classes eight-fold as large. Instructors deployed both traditional lecture with active learning elements and flipped-classroom techniques, in which students almost exclusively engage in active learning during class periods. Computer Science, which at JHU is an engineering department that sits fully within the Whiting School of Engineering, faces the highest instructor to student ratio in its introductory offerings and copes with this by providing optional supplementary coursework during which students engage in paired-programming (Braught, Wahls, & Eby, 2011) exercises under the close supervision of teaching assistants.

In the prior work undertaken by Magana and Falk, a "worked-out example" methodology was deployed and evaluated as a means to mitigate the issue of cognitive overload discussed in the prior section (Vieira, Magana, Roy, Falk, & Reese, 2016b; Vieira, Yan, & Magana, 2015). In the most successful deployment, students were provided with access to step-by-step examples of sample solutions to programming problems. These solutions included conceptualization of the problem, algorithm development, and programming resulting in a final working, but uncommented, code. In our work we identified that a very effective strategy to engage students in the understanding of worked-out examples was by having them write in-code comments (Vieira et al., 2017). Students were required to comment this code for the first four such examples as a strategy to self-explain such examples. These were graded. Subsequent worked examples could be commented for extra credit. Students reported such practices to be useful for their learning.

III.C Sequencing

Sequencing plays a key role in structuring student learning by providing experiences that support the acquisition of knowledge and its deployment by embedding it within clear and, in some instances, familiar contexts and then gradually removing these contextual supports. In this way sequencing can provide stepping stones toward increasing adaptive expertise by encouraging students to deploy their new knowledge in successively less structured and more unfamiliar contexts. For instance, object scaffolding (Hensel & Sun, 2006), has been described as a pedagogical method in which current student learning is anchored by a conceptual map derived from previous learning (Sticklen, Amey, Eskil, Hinds, & Urban-Lurain, 2004). Object scaffolding has been proposed together with a learning progression that builds on familiar concepts by beginning with scalar operations and then moving to vector and array operations (Sticklen et al., 2004). Similarly the use-modify-create approach ([NRC], 2011; Malyn-Smith &

Lee, 2012), has been identified as a potential strategy to introduce computational practices to novice learners.

In our own prior work (Magana et al., 2013) we considered a course developed using the *How People Learn* [HPL] framework (Bransford, Brown, & Cocking, 2000) that emphasizes, amongst other dimensions, knowledge-centered and learner-centered instructional practices. In that course a sequence was adopted in which in-class exercises would proceed from students being presented with a working code to read and execute. Next students were provided with a code, but that code was either malfunctioning, incomplete or required extension. The final exercise involved developing a code to meet a specification that was at least conceptually related to the first two codes. The course as a whole was observed to result in high student self-assessments of knowledge of programming, utility of computation and intention to continue to pursue computing opportunities in their studies and careers.

A different sequencing strategy was deployed in later research regarding the same course to help students structure their work on complex projects (Vieira, Roy, Magana, Falk, & Reese, 2016). During the initial problem recognition phase, the student works to understand the problem and create a plan to work toward a solution. The student uses verbal and mathematical representations for this purpose. In the second phase, called problem framing, the student executes the plan to create a solution in the form of an algorithm instantiated as a program. Finally, in the problem synthesis phase, the student completes the plan by evaluating the solution according to both instructor-provided and student-generated criteria. The implementation of test cases is one of the most challenging phases for students, and one of the most valuable for those who go on to programming practice in any context (Vieira, Magana, Roy, Falk, & Reese, 2015).

III.D Sociology

Preliminary work that focuses on integrating the sociology component has primarily described the way programming content has been combined with engineering problem solving (e.g., Azemi & Pauley, 2006; Devens, 1999; Hrynuk, Pennington, Illig, & Dempsey, 2008; Luchini, Colbry, & Punch, 2007; Morrell, 2007; Morris et al., 1996). Although this literature has documented instances of how computing knowledge has been combined with problem solving, information is very limited about the effectiveness of these implementations. Furthermore, this literature rarely builds theory on how learning about computing actually happens or how can it be effectively supported. Therefore, the sociology component is one of the least explored aspects for integrating computation in engineering education.

Our findings suggest that programming preparation grounded in anchored instruction provided an important foundation that goes beyond increasing students' control self-beliefs. Anchored instruction (The Cognition Technology Group at Vanderbilt [CTGV], 1990), is a cognitive perspective used to overcome the limitations of inert knowledge by providing sufficient contextual information for students to understand how computing gains utility and is operationalized in the disciplinary setting. This preparation seems to effectively enable students to leverage computational practices for the purpose of acquiring disciplinary concepts (Magana et al., 2016).

IV. Establishing Consensus via an "Organizational Development" Strategy

It is important to acknowledge that the above research and scholarly teaching effort was not happening in isolation but, rather, was being undertaken while many other independent factors were swiftly evolving in computing education within the Whiting School of Engineering at Johns Hopkins and, indeed, throughout the U.S. engineering education enterprise (Brown, 2016;

Dryfoos, 2017; Ghosh, 2017; Johnson, 2018; Shread, 2017). Most notable at Johns Hopkins was the fact that CPMSE was only one of a number of such courses that sprung up, nearly independently, in at least six of the other nine academic engineering departments. While the specific reasons for the development of such courses varied to some extent from department to department, common motivations included the perceived need for an earlier inculcation of students into computational practices, a desire to ground such practices within specific disciplinary contexts, and an intention to ensure that computational learning included a focus on data analysis that laid the groundwork for modeling and simulation skills. Later, in Section V, we will argue that related innovations were occurring in other departments simultaneous to our scholarly teaching effort due to a complexity leadership theory change strategy embedded in the JHU institutional structure. But we defer that argument for now to sustain, for now, a chronological narrative.

Multiple pressures on the computing curriculum across disciplines led the Whiting School of Engineering to take an organizational development approach to change. First, the demand for computing education at Johns Hopkins and many other universities ([NCES], 2017), was growing rapidly. Between 2012 and 2016 the number of Computer Science majors at Johns Hopkins increased by 125%, more than doubling and representing a growth rate of higher than 22% per annum, along with a similarly dramatic increase in Computer Science minors. In an extremely encouraging sign, many of the Computer Science majors were concurrently majoring in another engineering discipline, developing a cadre of students with the kind of cross-disciplinary knowledge necessary for computational tool building in the engineering context. However, this explosive growth was straining the teaching capacity within Computer Science at a time when Computer Science Ph.D.s were in such high demand in the workforce that hiring faculty in this discipline had become onerous for academic institutions (Metz, 2018). Second, the newly established computing courses in non-Computer Science disciplines did not all prepare students to pursue Computer Science at a deeper level, thus adding to the requirements for students pursuing joint study that combined engineering and Computer Science.

In September 2016, in light of the rapidly changing demands for computing education, the dean of the Whiting School of Engineering charged an ad hoc committee, chaired by Falk and faculty from five of the ten engineering departments including Computer Science to report on the outlook for undergraduate computing education within the school. The faculty on this committee had all taught, and a number had developed, introductory computing classes. The formation of this committee initiated an *Organizational Development* change strategy. Such a strategy is characterized by a more top-down approach in which, according to Borrego and Henderson (2014), "planned changes (interventions) in work settings are intended to create cognitive changes in employees, which lead to behavioral changes and improved organizations." The committee issued a report in April of 2017 that contained three main recommendations. The first was to rationalize introductory computing into a common course that would serve the needs of students across all engineering departments. The second was to more proactively weave computing into the core engineering coursework. The third was to stand up a range of computing minors to supplement existing minors. These new minors would provide guidance for students regarding how to navigate the various computational offerings across departments. These would be supported by standing faculty committees drawing on faculty talent from across the school. The members of these committees would provide oversight, advise students, and ensure that the critical coursework was routinely offered. The full report of the committee can be found in an online archive (Falk et al., 2017).

The ad hoc committee met approximately twice per month to undertake an open-ended discussion about the challenges departments face with respect to computing education. Notes were taken during the meeting and shared on a projector so that all critical input was captured accurately. These notes were also shared with all participants. A number of challenges were identified, one of which was the lack of a unified introductory computing course for all engineering majors. In an effort to assess the difficulty in aligning the educational goals embedded in all such courses offered across engineering, the committee considered the similarities and differences in educational approaches and learning objectives in the various courses that had emerged across the school. An effort was made to enumerate all curricular topics covered in these courses, and to abstract from these master learning objectives considering all such classes. These were:

1. Achieve familiarity with the essential computer programming concepts and techniques used by engineers.
2. Understand and apply the basic concepts and principles of structured programming and object-oriented programming.
3. Enact an iterative development process that includes problem conceptualization, design of algorithms, formulation of test cases, incremental coding, intermediate and final testing and debugging in order to produce a working solution.
4. Gain experience translating a well-posed engineering problem, as described by a mathematical representation, into a functioning program in order to analyze or solve the problem.
5. Develop sufficient expertise in computational thinking, deployment of computational tools, and utilization of data structures and common algorithms to pursue more advanced study in computing.

The curricular topics were then revisited to assess which topics were essential to any unified course, termed "Gateway Computing," that would meet these master learning objectives. Other topics were termed optional, in that they could be included in Gateway Computing, but were not essential to meeting the master learning objectives.

The committee's work aligned with the Organizational Development Logic Model presented by Borrego and Henderson (p. 238). The discussions that focused on the learning objectives necessary for students to be prepared both to incorporate computing work into their engineering studies, and to potentially pursue deeper expertise in Computer Science were crucial for building consensus between departments. A general consensus emerged that the computer language in which such skills were to be taught was a secondary consideration, as was the particular engineering context in which students experienced how computing found application within engineering. It was decided that sections would be taught in multiple computer languages and engineering contexts with first-preference of enrollment to a section provided to students enrolled in the major of the application area. The committee also made, in total, three master recommendations for change detailed in the report, the first of which related to introductory computing courses. The report articulated in detail the vision, guiding assumptions, and principles of the logic model described in Borrego and Henderson, which was adapted from prior literature on organization development and transformation (Porras & Silvers, 1991).

Subsequent to the release of this report, in July 2017, Falk assumed the title of Vice Dean for Undergraduate Education within the engineering school. This provided additional direction of effort and a clear platform from which to pursue school-wide reform regarding computing education. An information campaign was launched to make a strong case to departments that

there were benefits to be gained by joining the effort to implement this new approach to introductory computing instruction. This involved presentations at chairs meetings and at individual departmental faculty meetings. The reception in most departments was cautiously enthusiastic, demonstrating that cognitive change had begun. As noted in the logic model, inducing cognitive change is a precursor to behavioral change. No department was coerced to participate. In consultation with colleagues from the involved departments, several implementation decisions were made. In keeping with the report recommendations, it was decided to structure the class into small sections in order to lower the barrier for active learning strategies and in the hopes of making flipped-classroom implementation the norm. A hiring committee was established to recruit new instructors to make the small-section implementation possible. Two teaching faculty were hired in a 50:50 partnership between the dean's office and the departments that most closely aligned with the expertise of the applicants. Self-paced assessments were embraced as a way to provide students who enter with various levels of familiarity with computer programming to achieve similar learning goals while mitigating the stress such environments can cause for students with less prior exposure to the subject material (Offutt et al., 2017). Perhaps most notably, the establishment of a community of practice was seen as critical for sharing best practices between instantiations of this course and for pooling expertise in Computer Science and in disciplinary engineering application, i.e. Computational Science and Engineering.

As noted above, Falk's initiation into this work was originally informed by a *Scholarly Teaching* approach. This experience led Falk to encourage the Gateway Computing committee to collect data to assess the impact of the changes as the broader project transitioned to an *Organizational Development* approach. The hope was that collecting and analyzing data would convince skeptics that the curriculum redesign was worth adopting. During the first semester of implementation, sections of introductory computing will be taught with the traditional lecture method while others will employ the redesigned curriculum. Continuing to teach sections with a lecture approach will provide a control in the analysis. The CS1 concept inventory will be used to measure student learning differences between sections. It is a validated instrument that assesses student mastery of computing concepts typically taught in introductory courses (Tew & Guzdial, 2010). The team measured students' self-beliefs about computing utility and intention through an end-of-semester survey. This data was an extension of the educational research work published by Falk, Purdue, and Reese that was funded through the National Science Foundation Research Initiation and Research in Engineering Education Grants. Student general reactions to the course were captured through an end-of-semester survey and standard course evaluations. The team will also employ end-of-semester focus groups to explore students' perspectives in more detail to inform future changes

IV. Leveraging "Faculty Learning Communities" to Advance Change

This last aspect of the effort indicated a clear engagement of a *Faculty Learning Communities Change Strategy* (Borrego & Henderson, 2014), characterized by the integration of evidence based practices in the classroom through the development of a community of practitioners. The underlying logic is that instructional changes will be brought about by groups of instructors who support each other's learning and reflection on their teaching. Ideally, this will be done with a target course (p 234). At the time of this writing faculty are actively collaborating to design the resulting coursework for a multi-section Gateway Computing course. An initial offering in the Java language led by faculty in the Computer Science and Electrical and

Computer Engineering departments will be offered to incoming first-year students in the Fall of 2018. This is anticipated to be expanded to an offering that also includes the departments of Biomedical Engineering, Mechanical Engineering, and Materials Science and Engineering in the Spring of 2019. In Fall 2019 faculty from Civil Engineering and Environmental Health and Engineering will join the teaching team.

"Participation in a faculty learning community increases awareness of different teaching and learning styles and broadens awareness of different cultures and disciplines." (p 234). This process of making faculty aware of instructional best practices is essential to the future success of the course, and purposefully embedded into the design of the working group. The core Computer Science faculty are acting as project leads with the support of instructional designers and educational technologists. Together they conduct initial research on resources (e.g., interactive textbooks, delivery platforms) and develop draft instructional documents (e.g., curriculum maps, learning objectives, and assessment strategies) that are then reviewed and discussed with the extended faculty team that will adopt the curriculum model in subsequent semesters. This process of gaining consensus of the broader group after the core group drafts initial plans provides an excellent opportunity for peers to learn from each other and learn about cultural differences and program requirements across disciplines.

As we have done in the past, we have intentions to study the effectiveness of this hybrid change strategy that involves both top-down organizational leadership and faculty communities focused on the implementation of research-based instructional methods. We want to ask questions about the affordances, positive or negative, that development strategies such as faculty learning communities provide practitioners when situated in the context of enactment strategies, such as the organizational development approaches described above. To do this we plan to follow design-based research procedures and principles.

V. "Complexity Leadership Theory" as a Precondition for Innovation

In reflecting on the processes and confluences that have led to the place Johns Hopkins finds itself at the present moment, it seems appropriate to ask, "What preconditions set the stage for this curricular change upon which we are embarking?" Perhaps the most essential aspect of this has been the way our particular school, a highly research-focused and decentralized engineering school within similarly research-focused and decentralized university has manifested change. In our structure departments have almost total control over evolutionary changes in curriculum. School-level and higher approval is only required if new programs are created in the forms of majors or minors, or if a program undergoes substantial revision involving more than 1/3 of the programmatic content. This is an exceptional level of departmental autonomy that is deeply woven into the culture of the school. Even so, it is not without its detractors within the institution itself, and efforts have been made to create oversight boards and committees to vet curricular change on a more fine-grained level. After all, providing such autonomy creates tension between departments, as shifting requirements within one department can create demands on others. This requires, under optimal conditions, the maintenance of a level of cordiality and communication directly between departmental leaders or, under sub-optimal conditions, the intervention of upper administration at later stages once problems have become sufficiently critical to be brought to the deans. It also limits autonomy of upper-administrative leaders since processes require buy-in so as not to violate the organization's cultural norms. A case in point here being the need, described above, to engage departments voluntarily rather than instituting change by fiat. Careful implementation becomes essential to achieving full buy-in as

those departments who are not early-adopters gauge the disruption and advantages associated with a particular curricular change.

As noted before there were many stresses created by changes both by engineering departments standing up their own computational coursework, and by students' shifting choices of major and minor enrollment. These stresses created both the motivation for change at the same time that experiments undertaken autonomously by departments, like Falk's own scholarly teaching work, started to lay the groundwork for solutions to these stresses. In this way, the very decentralized nature of the university appears in retrospect facilitated the deployment of a *Complexity Leadership Theory Change Strategy* (Borrego & Henderson, 2014). In such a strategy patterns are disrupted, intentionally or due to outside forces, but the constant encouragement of novel approaches allows for the self-organization of solutions. In this context, the sense-making aspect of the committee report commissioned by the dean may be seen as an aspect of such a strategy, and the role of the ad hoc committee was primarily to act as a sense-maker in this instance. It seems likely that in the absence of the freedom provided by the de-centralized structure of the university, the ad hoc committee would not have benefited from the experiences brought by the faculty who had experimented with a variety of curricular innovations. Involving faculty actively in the generation and synthesis of new emerging ideas by affording freedom may have lowered the barriers to acceptance of the changes proposed by the committee.

The emergence of change agents to facilitate sense making and execution is also precondition for change in the complexity leadership framework. Falk has, in a number of ways, acted in the role of a change agent to execute this strategy. Coordinating the efforts of the Computing Curriculum ad-hoc committee and Gateway Computing working group created, "the conditions for productive emergence to occur and then to take the productive results from this process and ensure that these are integrated into the organization" (p. 242). Successfully navigating this role required a dual identity as both peer faculty and member of the Dean's Office when working with the committee and working group. Falk was a peer both disciplinarily and instructionally, in that he teaches an introductory computing course who previously employed many of strategies that informed the curriculum changes. Serving as Vice Dean of Undergraduate Education empowered him to enact these changes. In this role he was empowered to lead meetings of the department chairs to discuss and debate the recommendations, and to recruit faculty from their departments to implement the changes. As Vice Dean he was expected to understand the impact of these changes on the various constituencies including faculty, students and accreditors. The advising staff who support students in navigating the curriculum also report to him, and so he could play a coordinating role in ensuring that students could smoothly navigate these changes. He was also in a position to assign educational technologist and instructional designers to support the faculty who would develop the new computing courses.

While it would be interesting to study the complexity leadership aspects that constitute the preconditions of this curricular change, this would require comparing the approach and outcomes to other institutions with differing organizational structures. It would be particularly interesting to compare this instance within the Whiting School of Engineering at Johns Hopkins to changes at institutions that impose more restrictive processes for vetting curricular changes or that impose top-down changes without a process of recruitment that allows for departmental autonomy.

VI. Conclusions

Effecting innovation in how our institutions of higher education undertake their core mission of educating students is difficult and messy work. By articulating a clear theoretical framework with which one can classify and assess strategies, institutional changes can potentially be undertaken with greater chance of success and minimal disruption. Research-based educational innovations are one example of opportunities for innovation that are highly complex, involving changes in the cognitive and behavioral patterns that govern our institutions. These innovations often fail to spread because our established cultural practices around teaching are so entrenched that motivating faculty to adopt more effective strategies can require commitments of time, energy and personal and professional development that come at a perceived professional cost. At the same time, most faculty take deep pride in their teaching and their commitment to the educational mission. As we show here, finding ways to bring these cultural values into play and lowering barriers to enacting research-based educational methodologies requires not a single strategy, but layers of strategies that can flow, one into the other.

Here we paint a picture where providing a de-centralized organizational structure within which faculty have the freedom to experiment with novel educational approaches serves as an enabling precondition in line with the complexity leadership theory-based strategy. In such contexts, where encouraged, scholarly teaching strategies can emerge wherein faculty take responsibility for measuring the effectiveness of the changes they are implementing. If leveraged by the university, such innovative initiatives can provide opportunities for shifting to an organizational development strategy, building consensus toward cognitive and behavior changes to be enacted across broader sectors of the university. It is our belief, and intention to show, that these efforts can then be disseminated effectively via a faculty learning communities strategy that engages faculty within a community of practice. In this paper, however, we describe several explanations for how curricular change originated and propagated through the school,

Certainly, this particular sequencing of strategic approaches is not unique and we have no reason to believe that it is even ideal. It is our hope, however, that a description of the shifts in strategy we have participated in at our own institution is helpful for others conceptualizing how to approach the problem of educational innovation via research-based methods in their own institutions of higher education.

Acknowledgments

The authors acknowledge support from the National Science Foundation Division of Engineering Education and Centers under Grant No. 1137006, Grant No. 1329262, and Grant No. 1449238.

References

[ABET]. (2013). *2014-2015 Criteria for Accrediting Engineering Programs*. Retrieved from Baltimore, MD: http://www.abet.org/uploadedFiles/Accreditation/Accreditation_Step_by_Step/Accreditation_Documents/Current/2014_-_2015/E001%2014-15%20EAC%20Criteria%203-13-14(2).pdf

[ASEE]. (2013). Transforming Undergraduate Education in Engineering (TUEE) *Phase I: Synthesizing and Integrating Industry Perspectives*. . Arlington, VA: American Society for Engineering Education.

[NCES]. (2017). Digest of education statistics. *National Center for Education Statistics.* Retrieved from https://nces.ed.gov/programs/digest/d17/tables/dt17_322.10.asp?current=yes

[NRC]. (2011). Report of a workshop on the pedagogical aspects of computational thinking *National Research Council*. Washington, D.C.: National Research Council of the National Academies.

Azemi, A., & Pauley, L. (2006). *Teaching the introductory computer programming course for engineers using Matlab and some exposure to C*. Paper presented at the Proceedings of the 2006 American Society for Engineering Education Annual Conference & Exposition.

Barab, S., & Squire, K. (2004). Introduction: Design-Based Research: Putting a Stake in the Ground. *The Journal of the learning sciences, 13*(1), 1-14.

Borrego, M., & Henderson, C. (2014). Increasing the use of evidence-based teaching in STEM higher education: A comparison of eight change strategies. *Journal of Engineering Education, 103*(2), 220-252.

Bransford, J., Brown, A. L., & Cocking, R. R. (2000). *How People Learn: Brain, Mind, Experience, and School (Expanded Edition)*. Washington, D.C.: National Academy Press.

Braught, G., Wahls, T., & Eby, L. M. (2011). The case for pair programming in the computer science classroom. *ACM Transactions on Computing Education (TOCE), 11*(1), 2.

Brown, J. (2016). Interest in computer science surges. Enrollment is way up, and more women are signing on. *BU Today*. Retrieved from http://www.bu.edu/today/2016/computer-science-enrollment-increasing/

Cobb, P., Confrey, J., diSessa, A., Lehrer, R., & Schauble, L. (2003). Design experiments in educational research. *Educational researcher, 32*(1), 9-13.

The_Cognition_Technology_Group_at_Vanderbilt. (1990). Anchored instruction and its relationship to situated cognition. *Educational Researcher*, 2-10.

Collins, A., Brown, J. S., & Newman, S. E. (1989). Cognitive apprenticeship: Teaching the crafts of reading, writing, and mathematics. *Knowing, learning, and instruction: Essays in honor of Robert Glaser, 18*, 32-42.

Devens, P. E. (1999). *MATLAB & freshman engineering*. Paper presented at the Proceedings of the 1999 American Society for Engineering Education Annual Conference & Exposition.

Dryfoos, D. (2017). With large classes and waitlists, CompSci feels growing pains. But are those pains unique to Duke? *The Chronicle.* Retrieved from http://www.dukechronicle.com/article/2017/10/with-large-classes-and-waitlists-compsci-feels-growing-pains-but-are-those-pains-unique-to-duke

Du Boulay, B. (1986). Some difficulties of learning to program. In E. Soloway & J. C. Spohrer (Eds.), Studying the novice programmer (pp. 283-299): Lawrence Erlbaum.

Falk, M. L., Beer, M., Budavari, T., Kraemer, D., Marra, S., & Selinski, J. (2017). Computing in the engineering core curriculum at JHU: Report of the ad hoc committee on computing education of the Johns Hopkins Whiting School of Engineering. *Johns Hopkins University.* Retrieved from https://docs.google.com/document/d/1o7EuBvLvlXW5FaE29XerF0EG6DYsObmXTy6II8jSb_o/edit#heading=h.kws498uuvf1c

Ghosh, A. (2017). Pollack addresses increasing enrollment in computer science classes. *The Cornell Daily Sun.* Retrieved from http://cornellsun.com/2017/11/07/pollack-addresses-increasing-enrollment-in-computer-science-classes/

Hatano, G., & Inagaki, K. (1986). Two courses of expertise. In H. Stevenson, J. Azuma, & K. Hakuta (Eds.), *Child development and education in japan* (pp. 262-272). New York, NY: W.H. Freeman & Co.

Henderson, C. Finkelstein, N., & Beach, A. (2010). Beyond dissemination in college science teaching: An introduction to four core change strategies. *Journal of College Science Teaching*, 39(5), 18.

Henderson, C. Beach, A.L., & Finkelstein, N. (2011). Four categories of change strategies for transforming undergraduate instruction. *Transitions and transformations in learning and education*, 223-245.

Hensel, R., & Sun, Y. (2006). *Application of object scaffolding to develop a hands-on problem-centered, and project-based freshman MATLAB course*. Paper presented at the Proceedings of the 2006 American Society for Engineering Education Annual Conference & Exposition.

Hrynuk, J., Pennington, M., Illig, D., & Dempsey, J. P. (2008). *Freshman engineering:an introductory computer course teaching MATLAB and LABVIEW*. Paper presented at the Proceedings of the 2008 American Society for Engineering Education Annual Conference & Exposition.

International_Engineering_Alliance. (2011). *Washington Accord Program Outcomes*. Retrieved from Washington, DC: http://www.ieagreements.org/Rules_and_Procedures.pdf?4504

Johnson, S. (2018). Computer science degrees and technology's boom-and-burst cycle. *EdSurge.* Retrieved from https://www.edsurge.com/news/2018-04-03-computer-science-degrees-and-technology-s-boom-and-bust-cycle

Lister, R., Adams, E. S., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., . . . Seppälä, O. (2004). A multi-national study of reading and tracing skills in novice programmers. *ACM SIGCSE Bulletin, 36*(4), 119-150.

Luchini, K., Colbry, D., & Punch, W. (2007). *Designing introductory programming courses for graduate and undergraduate students: a parallel case study*. Paper presented at the Proceedings of the 2007 American Society for Engineering Education Annual Conference & Exposition. .

Magana, A. J., & Coutinho, G. S. (2017). Modeling and simulation practices for a computational thinking-enabled engineering workforce. *Computer Applications in Engineering Education, 25*(1), 62-78. doi:10.1002/cae.21779

Magana, A. J., Falk, L. M., & Reese, J. M. (2013). Introducing Discipline-Based Computing in Undergraduate Engineering Education. . *ACM Transactions on Computing Education., 13*(4), 1-22.

Magana, A. J., Falk, M. L., Vieira, C., & Reese, M. J. (2016). A case study of undergraduate engineering students' computational literacy and self-beliefs about computing in the context of authentic practices. *Computers in Human Behavior, 61*, 427-442.

Malyn-Smith, J., & Lee, I. (2012). Application of the Occupational Analysis of Computational Thinking-Enabled STEM Professionals as a Program Assessment Tool. *Journal of Computational Science Education, 3*(1), 2-10.

McCracken, W. M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y. B. D., . . . Wilusz, T. (2001). A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. *ACM SIGCSE Bulletin, 33*(4), 125-180.

Metz, C. (2018). Facebook adds A.I. labs in Seattle and Pittsburgh, pressuring local universities. *The New York Times.* Retrieved from https://www.nytimes.com/2018/05/04/technology/facebook-artificial-intelligence-researchers.html

Morrell, D. (2007). *Design of an introductory MATLAB course for freshman engineering students*. Paper presented at the Proceedings of the 2007 American Society for Engineering Education Annual Conference & Exposition.

Morris, P. J., Long, L. N., Haghighat, A., & Brady, M. L. (1996). *Curriculum development in advanced computation*. Paper presented at the Proceedings of the 1996 American Society for Engineering Education Annual Conference & Exposition.

Narayanan, G. (2007). *Teaching of essential MATLAB commands in applied mathematics course for engineering technology*. Paper presented at the Proceedings of the 2007 American Society for Engineering Education Annual Conference & Exposition.

Offutt, J., Ammann, P., Dobolyi, K., Kauffmann, C., Lester, J., Praphamontripong, U., . . . White, L. (2017). *A Novel Self-Paced Model for Teaching Programming.* Proceedings from the Proceedings of the Fourth (2017) ACM Conference on Learning@ Scale.

Pea, R. D., & Kurland, D. M. (1983). On the cognitive prerequisites of learning computer programming.

Porras, J. I., & Silvers, R. C. (1991). Organization development and transformation. *Annual review of Psychology, 42*(1), 51-78.

Shread, E. (2017). CS enrollment continues to swell, department responds. *The Phoenix.* Retrieved from http://swarthmorephoenix.com/2017/02/16/cs-enrollment-continues-to-swell-department-responds/

Soloway, E., & Spohrer, J. C. (1989). *Studying the novice programmer*: Lawrence Erlbaum Hillsdale, NJ.

Stickel, M. (2011). *Putting mathematics in context: an integrative approach using MATLAB*. Paper presented at the Proceedings of the 2011 American Society for Engineering Education Annual Conference & Exposition.

Sticklen, J., Amey, M., Eskil, T., Hinds, T., & Urban-Lurain, M. (2004). *Application of object-centered scaffolding to introductory MATLAB*. Paper presented at the Proceedings of the 2004 American Society for Engineering Education Annual Conference & Exposition.

Tew, A. E., & Guzdial, M. (2010). *Developing a validated assessment of fundamental CS1 concepts.* Proceedings from the Proceedings of the 41st ACM technical symposium on Computer science education.

Vieira, C., Magana, A. J., Falk, M. L., & Garcia, R. E. (2017). Writing in-code comments to self-explain in computational science and engineering education. *ACM Transactions on Computing Education (TOCE), 17*(4), 17:01-17:21. doi:10.1145/3058751

Vieira, C., Magana, A. J., Roy, A., Falk, L. M., & Reese, J. M. (2015). *Exploring Undergraduate Students' Computational Literacy in the Context of Problem Solving.* Proceedings from the Proceedings of the 122th ASEE Annual Conference and Exposition, Seattle, Washington.

Vieira, C., Magana, A. J., Roy, A., Falk, L. M., & Reese, J. M. (2016a). Exploring undergraduate students' computational literacy in the context of problem solving. *Computers in Education Journal., 7*(1), 100-112.

Vieira, C., Magana, A. J., Roy, A., Falk, L. M., & Reese, J. M. (2016b). In-code comments as a self-explanation strategy for computational science education. *In Proceedings of the 123rd ASEE Annual Conference and Exposition*.

Vieira, C., Yan, J., & Magana, A. J. (2015). Exploring design characteristics of worked examples to support programming and algorithm design. *Journal of Computational Science Education, 6*(1), 2-15.

Wang, F., & Hannafin, M. J. (2005). Design-based research and technology-enhanced learning environments. *Educational Technology Research and Development, 53*(4), 5-23.

Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L., & Wilensky, U. (2014). *Defining Computational Thinking for Science, Technology, Engineering, and Math.* Proceedings from the 2014 American Educational Research Association Annual Meeting, Philadelphia, Pennsylvania.

Yaşar, O., Maliekal, J., Veronesi, P., Little, L., & Vattana, S. (2015). Computational Pedagogical Content Knowledge (CPACK): Integrating Modeling and Simulation Technology into STEM Teacher Education. *Research Highlights in Technology and Teacher Education 2015*, 79.