ISSN: 2167-1435
v30 n2572

Confronting the Issues of Programming In Information Systems Curricula: The Goal is Success

Jeffry Babb
jbabb@wtamu.edu
Computer Information and Decision Management
West Texas A&M University
Canyon, TX 79106 USA

Herbert E. Longenecker, Jr.
longeneckerb@gmail.com
School of Computing
University of South Alabama
Mobile, AL 36688 USA

Jeanne Baugh
baugh@rmu.edu
Computer and Information Systems Department
Robert Morris University
Pittsburgh, PA

David Feinstein
dfeinstein@usouthal.edu
School of Computing
University of South Alabama
Mobile, AL 36688 USA

Abstract

Computer programming has been part of Information Systems (IS) curricula since the first model curriculum. It is with programming that computers are instructed how to implement our ideas into reality. Yet, over the last decade numbers of computing undergraduates have significantly declined. In addition, high failure rates persist in beginning and even advanced programming courses representing losses of students to the anticipation of production of future professionals. Perhaps the main reason the current model curriculum has removed programming is to enable a higher degree of success with higher rates of program completion. Ironically, in the face of this decision, national skills expectations demand programming abilities from graduates of computing programs. Further, most all business schools require programming, and all ABET-accredited programs have multiple courses in programming. While there are challenges in a programming sequence, there is evidence that multiple approaches can be taken to improve the outcomes and perception of success. There is the perception that the problems with this sequence will be improved significantly.

Keywords: programming, class performance, outcome improvement, curriculum, skills achieved

1. INTRODUCTION

Information Systems model curricula - IS'90 (Longenecker and Feinstein, 1991), IS'95 (Couger, et al 1995), IS'97 (Couger, et al 1997), and IS2002 (Gorgone, et al, 2002) – have stated that a principle focus of these curricula was to produce graduates who are competent and confident in developing and deploying Information Systems. These exit-level goals have pervasively guided and shaped the development of these model curricula. The skills necessary to achieve these goals were identified by a survey of faculty and practitioners (Landry et al, 2001), and were reaffirmed by the work of Colvin (2008) based on surveying graduates 3-5 years out of school.

Interestingly, Longenecker, Feinstein, Babb and Waguespack (2013) have demonstrated that the skills expected of IS practitioners have not changed over this time, despite current trends in industry. In fact, the Department of Labor expectations for IS-related STEM-oriented jobs mirrors the expectations of the earlier curricula (Longenecker, Feinstein and Clark, 2013). Clearly, the marketplace expects and demands that our graduates possess technical skills such as a command of programming and ability to interoperate with databases.

Furthermore, the bulk of business schools with IS programs believe in the necessity for programming and database skills (Apigian and Gambill, 2012); approximately 99% of these schools offer at least one programming course and all offer at least a course in database. Likewise, the 47 ABET-accredited IS programs offer multiple programming courses as well as database (Feinstein, Longenecker, Shrestha, 2013).

In the IS 2010 (Topi, et al, 2010) model curriculum, programming was omitted from the list of requirements for an IS degree and relegated as being optional. Given the response of business schools and ABET accredited programs, it can be difficult to understand the reasoning being the omission of programming in the IS 2010 model curriculum. Since the "Dotcom" bust of the early 2000's, the number of students showing up for IS degree programs has decreased significantly; many programs have disappeared. Given that programming is difficult and there is a high degree of failure of students in these classes, it is not surprising that IS 2010 designers withheld programming as a requirement.

Ultimately, as programming is important, and there are difficulties in teaching/learning the skill, then effort must be spent in mediation of the difficulties. Some of these methods will be addressed within this paper. We hold that a few things are fairly certain: the need for computing professionals will remain high; that computing is a diverse field, where room exists for information systems as a discipline; and, that ABET's Computing Accreditation Commission (CAC) is correct in requiring that all computing disciplines share a core concern in learning about programming. As the spectrum ranges from concerns about the machine up to individual and organizational needs (Shackleford, 2006), programming remains a "lingua franca" as a means for all computing professionals to understand how data and information continue to transform our world.

As we argue for renewed effort for IS educators to remain grounded in the fundamentals of computing by holding fast in our commitment to instruction in computer programming, we make our case as follows. First, we present evidence that programming has appropriately remained at the core of the IS curriculum as a requisite skill for our graduates. We next delineate and explicate what we understand as goals for the programming sequence in an IS curriculum. As this is a paper concerned with achieving success in teaching IS students how to program, we next discuss the various means by which students, educators, and programs fail in the delivery of programming instruction. We address these failures with a discussion of several cases and techniques which have garnered success. We then discuss the issue of achieving success in programming instruction and conclude with thoughts moving forward. Our ultimate aim is that information systems remains among the relevant computing disciplines which will deliver on the need for computing professionals.

2. CURRICULUM AND PROGRAMMING

Information systems curricula have existed for about fifty years (Longenecker, Feinstein and Clark, 2013). Programming and database have always been integral to these programs. With the exception of IS 2010 (Topi, et al, 2010), programming and database have expanded in their specification in our model curricula. A complete specification of the skills of these curricula including IS'90 (Longenecker and Feinstein, 1991), IS'95 (Couger et al, 1995; Gorgone et al, 1994), IS'97 (Couger et al, 1997; Davis et al, 1997a; Davis et al, 1997b), and IS 2002 (Gorgone et al, 2002a; Gorgone et al, 2002) have been synthesized and is presented in terms of skills specified and expected in IS curricula (see Tables 1 – 5). Recent evidence leaves no reason to suspect that the guidance in these past curriculum models has changed.

The skills presented in model curricula prior to IS 2010 are consistent with the department of labor specifications (see DOL1, 2010; DOL2, 2010; DOL3, 2010; and DOL4, 2010). The applicability of these specifications is reviewed by (Longenecker, Feinstein, Babb and Waguespack, 2013), and is compatible with the skills lists presented as Tables 1 – 6. Furthermore, Computerworld (Pratt, 2013) stated that 60% of new hires will be hired as programmers; our personal observations corroborate this.

Apigian and Gambill (2010) studied the academic catalog of 240 schools of business and found that 99.17% taught at least one course in programming. A study of ABET accredited programs shows that these programs required at least several courses in programming. Aasheim et al (2012) also found that there is an expectation of industry that students must know programming.

During the past decade there has been a decrease in the number of students in IS programs. In addition, in some programs as many as 70% of students fail to complete coursework in programming sequences. So the challenge of suggesting that IS programs require multiple courses in programming combined with the difficulty in successfully helping students to survive and thrive is an understandably hard sell. Yet, many jobs and careers which rely on programming are available. There is evidence that legislators are awakening to this reality: "Computer programmers are in great demand by American businesses, across the tech sector, banking, entertainment, you name it. These are some of the highest-paying jobs, but there are not enough graduates to fill these opportunities" (Marco Rubio,
Senator, Florida, http://www.code.org).

One recommendation for a model CIS curricula (Longenecker, Feinstein, Babb and Waguespack, 2013) suggests that three courses in programming with database as a prerequisite to the third course will be necessary for a successful two-course capstone sequence (Reinicke and Janicki, 2010). This capstone sequence will produce desired skills needed by the IS industry.

3. GOALS OF PROGRAMMING SEQUENCE

The goals of the programming sequence are clearest if the goals of the major, degree, and discipline are clear as well. For our purposes, the goal of a program in information systems is to develop professionals who are able to design, develop, implement, manage, maintain, and strategically and tactically use information systems. While we do not purport that each graduate will engage in all of these activities during their professional career, we do assert that such a foundation is optimal. Thus a foundation in analysis and design, data management, and application development are each essential.

Data Management
An information system's ability to consume, produce, and transform data and information is quite fundamental to its existence. Students must be versed in the means by which the computing devices – which constitute the information system – handle data. While this concern typically reduces to the study of relational database management systems (RDBMS), an understanding of data and information must extend into the realm of abstractions (such as the degree to which we synchronize systems analysis and design with Entity-Relationship Diagrams) and into the detailed realm of computing architecture (understanding how computers and operating systems work). Thus, while understanding operating systems, computer hardware, and the implementation details of software are all equally-important data management concerns, RDBMSs remain a central concern for data management as RDBMSs are engineered to handle the myriad concerns of data and

information.

Among reasons that we include "database" in the curriculum is that we recognize that understanding the techniques and knowledge associated with RDBMSs remain critical to information systems. Also, instilling within students an awareness of the tight relationship between data and logic is also required.

The topics in Table 7 are offered as being critical to an understanding of how and RDBMS works. Many of the features and aspects of an RDBMS underscore the requirement that data management is governed by both business logic and internal integrity logic (hence the presence of "stored procedures" as an extension to many RDBMS product).

Programming

Few would argue that information systems students should study both systems analysis and data management. In fact, perhaps some of the most "management"-leaning information systems programs generally retain these topics. What we hope to demonstrate is that each are not only tied to programming, but both are dependent on programming.

As this paper is about success with teaching information systems students programming – a success we hold as essential and non-optional – we offer a detailed accounting of the knowledge, skills, and competencies essential to facilitating student success. However, we bound this success by framing the material against an overarching goal: that students are prepared to achieve success in a comprehensive, immersive, and applied capstone course which focuses on the design, development, testing, implementation, and management of an information system.

Toward this end, we envision and present Table 8, the sequence of programming topics which lead to success in the capstone course. Table 8 is a list that is reasonably representative of the major milestone concerns for the programming topic for information systems students. It is a list that surely must be broken into multiple course receptacles, which may not be possible for all programs. However, we propose that the topics in the list are requisite for students to be able to successfully enter into a capstone experience as described by Reinicke and Janicki (2010).

Towards Success

While our proffered list of programming knowledge and topics is meant to serve as a viable list, it is debatable if it serves as a minimal (or even optimal) path in support of the capstone course. And, while the capstone is not the sole aim of this paper, the goals and intent of the capstone justifies the extent to which list is useful. Thus, for students and programs alike, we must distinguish between overarching goals towards ultimate mastery and goals that are achievable.

Students must be made aware of the overarching goals that lead to mastery as such a target is a healthy for our discipline. However, during the short time that students are in our care, we must also remain cognizant of goals that are achievable. It is clear that we can't achieve mastery, in most cases, during the short time that students are in the major. After university and college requirements have been met, many programs face a serious deficit in the number of credit hours that can used towards the degree, let alone programming.

Programs in information systems will necessarily have to scale their programming sequence to remain consistent with their capstone sequence. Our full list of programming competencies would likely require three courses in programming, and some programs, as currently designed, may not be able to accommodate this.

Getting the Balance Right

Action required for getting over the "hump" that many students experience while learning programming means successfully engaging at least the first 15 steps in Table 7. Regardless of the depth which a program in information systems can handle with respect to available credit hours, it is likely that those first 15 items should/would be covered. It is with these first steps that we should focus on the aspects of mastery that develop automaticity (or muscle memory). We use the term "mastery" in the sense that it is ultimately an inclination towards an endeavor which requires immersion, engagement, and tenacity. Or, simply stated: hard work. Our students will, step-wise and incrementally, engage in the persistent and iterative pursuit of programming (topics 1 - 15) in a manner that benefits from the transformative benefits of mastery (gaining in levels of competence and confidence).

Coaching
Steps 1 through 15 in Table 7 (and programming in general) require the application of effort that is well suited to a coaching methodology. Coach John Wooden, the winner of 10 championship NCAA games utilized a concept of a "pyramid of success" (2005). His attitude of success could easily become a goal for programming education: *"What was under my control was how I prepared myself and our team. I judged my success, my 'winning', on that. It just made more sense. I felt if we prepared fully we would do just fine. If we won, great; frosting on the cake. But at no time did I consider winning to be the cake… It's true everywhere in life. Hard work is the difference. Very hard work."* (Wooden and Jamison, 1997)

4. KNOWN FAILURE MECHANISMS

The literature on pedagogy, computing pedagogy, and IS pedagogy provides myriad failure mechanisms known to stymie student success. We discuss several categories of these failures here. There are failures related to the mode and delivery from the faculty: lecturing; lack of hands-on experience; lack of student follow-up (in the case of absences in particular). Some failure mechanisms are related to the situation of programming within the curriculum or with the structuring of the course. In particular, a programming course may lack the correct pre-requisites. In other cases, there is a lack of continuity between the courses in the sequence. Another known failure mechanism is the lack of proper teamwork structures that encourage team and peer-driven learning. A last category of failure is that of leadership, motivation, and correction. While students may lack maturity of may encounter issues related to their ability to perceive, receive, and manage failure, we can and must do more than chalk these issues up to being out of our control. The reader may consult Table 9 to see an elaborated of student failure issues.

Dealing with Lack of Student Maturity
While there is little doubt that students exhibit a paucity of maturity in many regards - failure to come to class; failure to have an attitude of success; immature reaction to our correction; returning a haughty response; lowering standards for everyone – it remains our obligation to sustain leadership and motivation to do our best to correct these behaviors. We are all aware that some students will wait until the last minute to try to complete a programming assignment. They need to be encouraged to begin the design of the solution as soon as the work is assigned. This type of behavior is seen in many college courses, not just a programming course, and is reflective of bad habits. This also shows a lack of maturity on the part of the student. As we are instructing for eventual automaticity in exercise of knowledge and skill in our students, we can also aim for automaticity of students' response such that maturity will become a default response.

Among the possible responses to student immaturity is to address students' transition into college (and the requisite maturing required for success) are "freshman seminar" or "first year experience" programs. Typical goals, learning outcomes, and requirements of such programs will include modules on academic success and responsible behaviors that lead to maturity. As we discuss failures, it is important to note that the imperatives underscored by these programs also extend into the context of teaching programming (and perhaps all of our courses).

Faculty Responsibility for Great Patience
It takes a faculty member with a great deal of patience to teach programming. Some faculty members feel that the student either has the ability to program or they do not. But the truth is that almost everyone can learn to program at some level. However, the faculty member must put in the time to help the students. Sometimes it requires a one-on-one session to review the programming code line-by-line with the student; showing him what he was doing wrong in his code. Coding mistakes made by a student are a great learning tool; however, an instructor must explain those mistakes and help the student to understand why the corrections are needed. It provides an insufficient learning experience to simply mark an assignment wrong without feedback. However, it may be more important to follow-up immediately on an absence from class, particularly one that occurs during the early phase of a course. This is so as the path to mastery requires constant engagement.

5. SUCCESSFUL APPROACHES

There are many different approaches to teaching a programming course. According to Wang (2010), programming requires thinking with abstract concepts, which is difficult for novices. Second, programming includes many different tasks, such as problem solving, algorithm and data structure design, programming language

ISSN: 2167-1435
v30 n2572

comprehension, testing, and debugging (Wang, 2010).

Table 10 presents a number of approaches that have been utilized to facilitate successful results to learning programming.

Speed of the Course
There are always a few students who will not be able to understand the topics in the course. This can be said of any course in a college curriculum. The marginal student may need additional actions taken to facilitate their success, such as extra help sessions, personal guidance during office hours, and tutoring sessions offered by the educational institution.

Actually it has been the experience of these authors that some students often want to create additional functionality in their programming assignments beyond what is required. Having some flexibility in the course topics allows the instructor to facilitate both the struggling student and the more advanced one.

Everyone Can Do IT
We all need to adopt a new attitude: everyone can do it. The web site http://code.org supports a non-profit organization dedicated to promoting computer science (specifically computer coding) as a requirement for all students. Their vision states that "every student in every school has the opportunity to learn how to code." Additionally, the organization supports the view that "computer science and computer programming should be part of the core curriculum in education, alongside other science, technology, engineering, and mathematics (STEM) courses, such as biology, physics, chemistry and algebra." (code.org)

Leaders in all phases of industry and education advocate for all students learning to code. Bill Gates feels that "Learning to write programs stretches your mind, and helps you think better, creates a way of thinking about things that I think is helpful in all domains." (code.org)

6. DISCUSSION

Programming is a necessary skill. According the United States Bureau of Labor Statics (BLS) website (www.bls.gov), students with a computing degree can expect to earn a starting salary around $70,000 per year. The BLS lists Computer Science as the highest-paying college degree and forecasts computer jobs as growing at two times the national average (code.org). However, as can be seen in Figure 4, students are not going into the field in large enough numbers to match the projected need.

Also, according to BLS, Systems Analysts are one of the fastest growing professions (22% growth per year) and have added 120,400 jobs since 2010 with a salary of $77,000 per year. Analysts require programming skills in a business context. Programmer analysts still are an entry point to the analyst profession at a salary of $71,380 per year.

Computer and Information Scientists (see BLS) hold a doctorate degree and invent and design new technologies, and find new uses for existing technologies in business, medicine and other areas, and earn $100,660 per year.

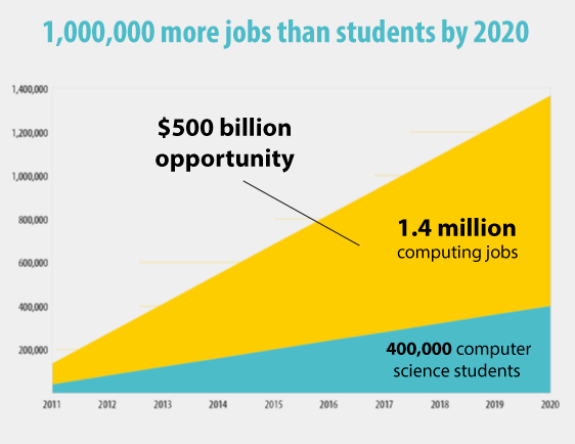


Figure 4 Job projections according to code.org

A programmer's job is to first solve problems. This entails a developing a detailed analysis of the problem, design of the solution (the computer instructions) and finally the test of the ultimate solution. Learning a programming language helps the IS students to understand how the computer actually works and processes instructions, not just how to use a computer. The student learns how to think logically and how to "tell" the computer what to do. As such, there are many different job opportunities for a student with an IS degree, many of which are not in coding. But with some experience in programming, the IS student will better understand things such as:

- what a file is and how it is accessed
- how an algorithm is used to solve a real life problem and how it is coded in the

computer world
- being able to understand some basic principles such as variable assignment and conditional branching

Problem-Solving Training

The skill set required for employment in the "real world" is constantly evolving. (Gallivan, Truex, Kyasny, 2004) Employers want their employees to program as well as communicate and document their work. Programming skills along with communication and ethical, interpersonal, and personal skills go hand in hand for successful employment (Aasheim, 2012). In fact, employers are increasingly demanding these competencies of their entry level employees (Gruba, Al-Mahmood, 2004). In this light, the imperative for success in our instruction in programming matters as programming provides, at the very-least an applied and useful means of developing a mental acuity for problem solving that is relevant for our times.

7. CONCLUSION

Many disagree about the importance of programming in an undergraduate IS degree. (Topi, Valacich, Wright, Kaiser, Nunamaker, Sipior, & de Vreede, 2010).

It is the opinion of others (Longenecker, Feinstein, Babb, and Waguespack, 2013), and these authors, that a programming course can be extremely beneficial to the IS student even if the student has no intention of writing code upon graduation. Todd Park U.S. Chief Technology Officer said "… technology and computers are very much at the core of our economy going forward. To be prepared for the demands of the 21st century—and to take advantage of its opportunities—it is essential that more of our students today learn basic computer programming skills, no matter what field of work they want to pursue." (http://www.code.org)

Writing a computer program is an exercise in logic. Since the basics of all computer work centers around these logical foundations, programming is fundamental to everything in computing. The computer can do nothing without a set of instructions (a program) to tell it what to do. A program can be anything from operation system instructions to instructions for a specific application, such as a game or a business process.

So, the issues at hand are straight forward and critically important. On one hand, the impact of programming can be seen daily in almost every aspect of our lives in personal technologic advances, in commerce and banking, and in health care and more; IS development requires great programmers and more of them (Pratt, 2013). On the other hand, teaching students programming remains very difficult. The incorporation of critical thinking, programming logic, technology and syntax must be blended in the process. Teachers of the discipline must not only be excellent with the skill, they must be attentive to the needs of young and beginning students. As educators begin to understand the coaching skills of John Wooden as applied with the technology of programming, the success of Wooden may translate into our own success. Of course we must define carefully and monitor progress, and make daily adjustments to the plan.

Also highly important is that students need to feel that the instructor is accessible. A study done by Yang and Cornelius (2004) also supports the concept that students must receive feedback on a timely basis. This is but one part of coaching.

While we call for very high demands, and illustrate very high stakes, at some point faculty will want to question the structure of our professional lives as academics – do we have the incentive and reward for the effort this will require? It is our position that we are dealing with an existential imperative. Either we provide value to the marketplace, or it will forget us. IS educators have a hard enough time to have to describe and explain that we exist, why we exist, and that we play a role and complement among the computing professions. To reverse on fundamental computing skills is to only increase our burden.

Faculty can, and must, be heroes and coaches. There is nothing more rewarding than hearing the student say that he never imagined that he could write a program such as the one he was turning in at the end of a semester. The educational institution must recognize not only how important programming is, but also how time-consuming it is for the instructor who teaches such a course. Successfully teaching programming does require more effort on the instructor's part (Escalente, 2008). A great deal of student follow-up needed for the beginners. Because faculty may spend a lot of extra time helping the first time programmer, it is essential

that the institution provide support in areas such as class size, tutoring and course release for extra office hours or help sessions. If the support is not there, the success of any such course cannot be assured (Gopalakrishnan, 2006).

As IS educators, we must also accept that the reward for the extra effort in the instruction of programming will not likely come in the form of financial compensation; the rewards will likely be cerebral and personal in nature. The feeling of walking out of the classroom knowing that the students "got it" is immeasurable.

The student who has never programmed before feels an enormous sense of accomplishment when the code all comes together and produces correct results. Students are very proud of their work. The Authors have taught programming for many years and from their experience it is clear that programming students show success in both the areas of increased technical skills and personal growth. In addition to the students' positive experiences, watching the transformation of the students from not understanding a simple output statement to writing a programming project encompassing many methods/functions is extremely rewarding to the instructor.

Once a student in a lab asked how the instructor could stand to go from computer to computer helping students with the same programming tasks. The answer to that question is very simple, it is what teachers do. We have little doubt that going these "extra miles" is normative behavior present in many institutions and also present in many who read this article. However, we illustrate the problem ensuring success in the teaching/learning of programming for our students as being profoundly important for the discipline. Maintaining a viable discipline is a concern we should all share. Lastly, the authors of this paper hold that a love and thirst for technology – learning it, teaching it, using it, studying it – should be in our blood. "Programming is exciting, stimulating, fun and develops new ways of thinking". (http://www.code.org) We, as instructors, are charged with helping to successfully prepare our students for the digital future.

8. REFERENCES

Aasheim, C., Shropshire, J., Li, L., Kadlec, C. (2012). Knowledge and Skill Requirements for Entry-Level IT Workers: A Longitudinal Study, *Journal of Information Systems Education. Summer2012, Vol. 23 Issue 2,* p193-204

Apigian, C.H. and Gambill, S.E. (2010). Are We Teaching the IS2009 Model Curriculum? *Journal of Information Systems Education, Vol. 21(4),* p411-420.

BLS (Bureau of Labor Statistics) (2013) retrieved July 21, 2013 at http://www.bls.gov/ooh/computer-and-information-technology/

Barki, H. and Hartwick, j. (1989). Rethinking the Concept of User Involvement, *MIS Quarterly,* 13(1)*,* 53-63.

Baugh, J. M. (2011). Make it Relevant and They Just May Learn it, *ISEDJ 9(7), December 2011.*

Baugh, J.M., Kohun,F. (2005) Factors that Influence the successful completion of a Doctoral Degree, *IACIS Conference Presentation*, Atlanta GA

Baugh, J. M., Kovacs,P. (2012). Large Programming Projects for the beginning programmer, *Issues in Information Systems,* 13(1), 85-93.

Baugh J. M., Davis G., Kovacs, P., Scarpino, J., Wood, D. (2009). Employers And Educators Want Information Systems Graduates To Be Able To Communicate, *Issues in Information Systems,* 10(1), 198-207.

Choobineh, J., & Lo, A. W. (2004). CABSYDD: Case-Based System for Database Design. *Journal of Management Information Systems, 21(3*), 281-314

Pratt, M.K. (2013). 10 hot IT skills for 2013, *Computerworld,*
http://www.computerworld.com/s/article/print/9231486/10_hot_IT_skill. Last Accessed: 07/15/2013.

Courte, J. and Bishop-Clark, C. (2005). Bringing Industry and Educators Together, *Proceedings of the 6th Conference on Information Technology Education, October* 20-22, pp. 175-178.

Colvin, R. (2008). Information Systems Skills

and Career Success, Masters Thesis, University of South Alabama, School of Computer and Information Sciences.

Cyrillo, M. (2011). Lean UX: Rethink Development, *Information Week, Nov. 14, 2011*.

Couger, J. D., Davis, G. B., Dologite, D. G., Feinstein, D. L., Gorgone, J. T., Jenkins, M., Kasper, G. M. Little, J. C., Longenecker, H. E. Jr., and Valachic, J. S. (1995). IS'95: Guideline for Undergraduate IS Curriculum, *MIS Quarterly* 19(3), 341-360.

Couger, J. D., Davis, G.B., Feinstein, D.L., Gorgone, J.T. and Longenecker, H.E. (1997). IS'.97: Model Curriculum and Guidelines for Undergraduate Degree Programs in *Information Systems, Data Base, 26(1), 1-94*.

Davis, G., J. T. Gorgone, J. D. Couger, D. L. Feinstein, and H. E. Longenecker. (1997). IS'97: Model Curriculum and Guidelines for Undergraduate Degree Programs in Information Systems. *ACM SIGMIS Database*, 28(1).

Davis, G.B., Couger, J. D., Feinstein, D.L., Gorgone, J.T. and Longenecker, H.E. "IS '97 Model Curriculum and Guidelines for Undergraduate Degree Programs in Information Systems," ACM, New York, NY and AITP (formerly DPMA), Park Ridge, IL, 1997.

DOL1 (2010). Summary Report for: 15-1121.00 - Computer Systems Analysts, retrieved from June 1, 2013.

DOL2 (2010). Summary Report for: 15-1141.00 - Database Administrators, retrieved from http://www.onetonline.org/link/summary/15-1141.00 June 1, 2013

DOL3 (2010). Summary Report for: 15-1132.00 - Software Developers, Applications, retrieved from http://www.onetonline.org/link/summary/15-1132.00 June 1, 2013.

DOL4 (2010). Summary Report for: 15-1134.00 - Web Developers, retrieved from http://www.onetonline.org/link/summary/15-1134.00 June 1, 2013.

Deperlioglu, O., Sarpkaya, Y., & Ergun, E. (2011). Development of a Relational Database for Learning Management Systems. *Turkish Online Journal Of Educational Technology - TOJET,* 10(4), 107-120.

Escalente, J. (2008). Jaime Escalente On Being a Teacher, retrieved July 21, 2013, at http://www.youtube.com/watch?v=FFMz8JRg8Y8

Feinstein, D.L., Longenecker, H.E., and Shrestha, D. (2013). A Study of Information Systems Programs Accredited by ABET In Relation to IS 2010, *Proceedings of ISECON, San Antonia.*

Foltz , C., Bryan, O'Hara, Margaret T., Wise, Harold, (2004). Standardizing the MIS course: benefits and pitfalls, *Campus-Wide Information Systems,* 21(4), 163 – 169.

Gallivan, M., Truex, D., and Kyasny, L., (2004). Changing patterns in IT skill sets 1988-2003: a content analysis of classified advertising, *ACM SIGMIS Database*, 35(36).

Gopalakrishnan, A. (2006). Supporting Technology Integration in Adult Education: Critical Issues and Models, *Adult Basic Education: An Interdisciplinary Journal for Adult Literacy Educational Planning*, 16(1) 39-56.

Gorgone, John T., J. Daniel Couger, Gordon B. Davis, David L. Feinstein, George Kasper, and Herbert E. Longenecker 1994. "Information Systems '95," *DataBase,* 25, (4), 5-8.

Gorgone, J.T., Davis, G.B. Valacich, J., Topi, H., Feinstein, D.L. and Longenecker. H.E. (2003). IS 2002 Model Curriculum and Guidelines for Undergraduate Degree Programs in Information Systems. *Data Base* 34(1).

Gruba Paul, Al-Mahmood Reem, (2004). "Strategies for communication skills development", ACE '04: *Proceedings of the sixth conference on Australasian computing education - Volume 30*

Kim, Youngbeom, Hsu, J., and Stern, M. (2006). An Update on the IS/TT Skills Gap,"

Journal of Information Systems Education, 17(4), 395-402.

Hunton, J.E. and Beeler J.D. (1997). Effects of User Participation in Systems Development: A Longitudinal Field Experiment, *MIS Quarterly,* 21(4) 359-388.

Hutchings, P. and A. Wutzdorff, 1988, "Experimental learning across the curriculum: Assumptions and principals." *New Directions for teaching and Learning, 35, 5-19*

Kazemian, F., and-T. Howles, 2008. Teaching challenges: Testing and debugging skills for novice programmers. *Software Quality Professional*, 11(1), 5-12.

Larson, S., Harrington, M.C.R. (2012). A Study of ABET Accredited Information Systems Programs in the USA, *2012 Proceedings of the Information Systems Educators Conference, New Orleans, Louisiana, USA,* p1,18.

Longenecker, H.E., Feinstein, D.L., Babb, J.S., and Waguespack, L.J. (2013). Is There a Need For a Computer Information Systems Model Curriculum?, *Proceedings of ISECON, San Antonio.*

Longenecker, H.E., Feinstein, D. L. and Clark, J. (2013). Information Systems Curricula: A Fifty Year Journey, *Information Systems Education Journal (ISEDJ) 11(6) December 2013.*

Markus, M. L. (1983). Power, politics, and MIS implementation. *Communications of the ACM, 26(6), 430-444*.

Mathews, J. (2010) Jaime Escalenti dies, inspired 1988 film 'Stand and Deliver', *The Washington Post, March 31, 2010.*

McCauley, R., S. Fitzgerald, 6. Lewandowski, L Murphy, B. Simon, L Thomas, and C. Zander. 2008. Debugging: A review of the literature from an educational perspective. *Computer Science Education* 18(2), 67-92.

Reinicke, B. A. and Janicki, T. N (2010). Increasing Active Learning and End-Client Interactions in the Systems Analysis and Design and Capstone Course, *ISEDJ 8(40) June 30, 2010.*

Robbert, M. A., Wang, M. Guimaraes, M., Myers, M. (2000). The Database Course: What Must Be Taught, SIGCSE *Bulletin Proceedings of 31st SIGCSE Technical Symposium on Computer Science Education, March*, pp. 403-404

Schwalbe, K (2010). Information Technology Project Management, Course Technology, Cengage, United States.

Seyed-Abbassi, B., King, R., & Wiseman, E. (2007). The Development of a Teaching Strategy for Implementing a Real-World Business Project into Database Courses. *Journal Of Information Systems Education, 18(3),* 337-343.

Shannon, L., Bennett, J.F., and Schneider, S. (2009). Cycle of Poverty in Educational Technology, *ISEDJ 7(71*) retrieved at http://isedj.org/7/71/

Shannon, L., Schneider, S. and Bennett, J.F.. (2010). Critical Think Measurement in ICT, *ISEDJ. 8(1),* retrieved at http://isedj.org/8/1/

Shannon, L. and Benette (2012). A Case Study: Applying Critical Thinking Skills to Computer Science and Technology, *ISEDJ*, 10(4), retrieved http://isedj.04g/2010-12/issn: 1545-696X.

Subramanian, N. and Whitson, G. (2010). Implementing a CNSS 4012 Certification in an Information Systems Curriculum, *Proceedings of the 14th Colloquium for Information Systems Security Education, Baltimore Inner Harbor, Baltimore, Maryland June 7 – 9, 2010*

Topi, H., Valacich, J., Wright, R.T., Kaiser, K.M., Nunamaker, J.F., Sipior, J.C., and Vreede, G.J. (2010). IS 2010 Curriculum Guidelines for Undergraduate Degree Programs in Information Systems, Association for Computing Machinery (ACM), Association for Information Systems (AIS)", retrieved July 14, 2012: http://www.acm.org/education/curricula/IS%202010%20ACM%20final.pdf

Wang, X. O. P. H. I. E. (2010). Teaching programming skills through learner-centered technical reviews for novice programmers.

ISSN: 2167-1435
v30 n2572

Software Quality Professional, 13(1), 22-28

White, G. (2012). Visual Basic Programming Impact on Cognitive Style of College Students, *ISEDJ 10(4)* retrieved at http://isedj.org/10/4

Wooden, J. R. and Jamison, J. (1997). Wooden, McGraw Hill, New York.

Wooden, J.R. and Carty, J. (2005). Coach Wooden's Pyramid of Success, Building Blocks of Life for a Better Life, Regal Books, Ventura, California.

Yang, Y, and Cornelius, (2004) "Students' Perception towards the Quality of Online Education: A Qualitative Approach", Association for Educational Communications and Technology, 27th, Chicago, Il. October 19-23 17 pp.

ISSN: 2167-1435
v30 n2572

Appendix

Table 1. Programming skills expected in model curricula (except IS2010)

Low level data structures	bits, bytes, number representation, money representation, character representation, rounding operations, overflow
Algorithmic Design, Data, Object and File Structures	analysis, design, development, debugging, testing, simple data structures (arrays, records, strings, linked structures, stacks, queues, hash functions). Functions, parameters, control structures, event driven concepts, OO design, encapsulation, classes, inheritance, polymorphism, sorting, searching
Problem Solving-identify problems, systems concepts, creativity	devise questions to help identify problems, apply systems concepts to definition and solutions of problems, formulate creative solutions to simple and complex problems, Fishbone-root cause, SWOT, Simon Model, Triz, ASIT; embracing developing technology; methodologies (waterfall, object, spiral etc.), dataflow, structured
Programming-principles, objects, algorithms, modules, testing	principles, concepts, control structures (sequence, selection, iteration); modularity, objects and ADTs, data structures, algorithmic design, verification and validation, cohesion, coupling, language selection, user interface design, desk checking, debugging, testing, error correction, documentation, installation, integration, operation; writing code in a modern programming language (e.g.,VB.net, Java, C#); interpreted and compiled computer languages; design tools; secure coding principles and practices
Application Development-requirements, specs, developing, HCI considerations	principles, concepts, standards; requirements, specifications, HCI planning, device optimization (e.g. touch screen, voice), development and testing, utilization of IDEs, SDKs, and tool kits; configuration management, installation, module integration; conversion, operation
Web page Development-HTML, page editors, tools	FrontPage, HTML, page building/edit tools, frames; http, Dreamweaver, Photoshop; Sharepoint, Joomla, Drupal, IDEs, SDKs, Snagit, Jing
Web programming-thin client, asp, aspx, ODBC, CGI, E-commerce, web services, scripting	Visual Studio; thin client programming: page design; HTML, *.asp/aspx coding; session variables / page security; ODBC; CGI programming; integration of multi-media; e-commerce models; tools: Java Script, Perl, Visual Studio, Java, Web services, XML server / client side coding, web services, hypertext, n-tier architectures; integration of mobile technology

ISSN: 2167-1435
v30 n2572

Table 2. Database and Data Management

Modeling and design, construction, schema tools, DB systems	Data modeling, SQL, construction, tools -top down, bottom up designs; schema development tools; desk-top/enterprise conversions; systems: Access, SQL Server/Oracle/Sybase, data warehousing & mining; scripts, GUI tools; retrieve, manipulate and store data; tables, relationships and views
Triggers, Stored Procedures, Audit Controls: Design / Development	triggers, audit controls, stored procedures, trigger concepts, design, development, testing; audit control concepts/standards, audit control Implementation; SWL, concepts, procedures embedded programming (e.g. C#)
Administration: security, safety, backup, repairs, Replicating	monitoring, safety -security, administration, replication, monitoring, repair, upgrades, backups, mirroring, security, privacy, legal standards, HIPAA; data administration, policies
Metadata: architectures, systems, and administration	definition, principles, practices, role of metadata in database design, repository, dictionaries, creation, ETL, administration, usage, tools
Data Quality: dimensions, assessment, improvement	Data Accuracy, Believability, Relevancy, Resolution, Completeness, Consistency, Timeliness; Data definition quality characteristics, Data model / requirements quality characteristics; Data clean-up of legacy data, Mapping, transforming, cleansing legacy data; Data defect prevention, Data quality employee motivation, Information quality maturity assessment, gap analysis
Database Security	SQL injection attacks and counter measures; encryption; limiting exposure in internet applications; risk management: attacks and countermeasures; Server Security management
Data sources and advanced types	Accessing external data sources; use of search engines; purchasing data; image data; knowledge representations
Database Server	Database server requirements, connecting from application to database, simple rules (no path to internet; local connection only), mounting and updating a database, use of script to enable application security; multi-user connections; replication and backup

Table 3. Personal, Interpersonal and Organizational Skills

Personal Skills-encouraging, listening, being Organized, principles of motivation	Having integrity, honesty; responsible attitude of personal responsibility; encouraging, listening, negotiating, being persuasive, being organized; Personality types and relationships (DISC, MBTI, COLOR)
Professionalism-self directed, leadership, time management, certification, conferences	being self-directed and proactive, personal goal setting, leadership, time management, being sensitive to organizational culture and policies; personal development (conferences, read literature, use self-development programs)
Professionalism-committing to and completing work	Persistence, committing to and rigorously completing assignments, can-do
Cognition	concepts of learning; sequential levels of learning (recognition, differentiation, use / translation, apply); relationship of learning and emotion
Mathematical Fundamentals	Mathematics (algebra, trigonometry, variables, operations, expressions, logic, probability, limits, statistics)
HCI Principles: underpinnings	Cognitive Process, education learning levels, interface design, concepts of usefulness, the 8 golden rules
Critical Thinking	fact recognition, argument strength, analysis (break into components), synthesis(assembling the components); abstraction; qualitative research principles
Individual behavior	learning styles (visual, auditory, kinesthetic), motor skills, linguistic mechanisms, auditory mechanisms
Communication-oral, written, multimedia, empathetic listening	oral, written, and multimedia techniques; communicating in a variety of settings; empathetic listening, principle centered leadership, alignment technical memos, system documentation, technical requirements; necessity for involvement; development of resistance
Develop Consultant Characteristics	build relationship, identify need, present alternatives, provide assistance as needed, make recommendations, be supportive
Ethics-theory/concepts, setting an ethical example	ethical theory and concepts, codes of ethics--AITP/ACM; setting an ethical example; ethical policies, intellectual property, hacking, identity theft
Learning to learn	journals, learning maps, habits of reading, listening to tape/cd, attending professional seminars, teaching others, meta-thinking, life long learning; human learning: recognition, differentiation, use, application, analysis, synthesis and evaluation
Teams-team building, vision / mission development, synergy building and problem solving; leadership	team building, vision and mission development, planning, synergistic consensus team leadership, leadership development, negotiation, conflict resolution

ISSN: 2167-1435
v30 n2572

Collaboration support by IT	It Solutions for Individuals and Groups, Problem solving mechanisms in support of meetings, consensus development
Impact of IT on Society	IT impact on individuals, on groups, on enterprises, on societies; knowledge work and support by IT; computer industry and society, work force requirements
IT Career Paths	Programmers, Application Developers, Information Analyst, Systems Analysis, Data Management, CIO, CTO

ISSN: 2167-1435
v30 n2572

Table 4. The Context of Information Systems

Learning Business Process and Environment	learning business process and environment, exchanges, competitive position, e-business, global concepts, business models, Creating value, Value chain, improving value creation; financial markets, determining value of securities; organizational models
Accounting, Distribution, supply chain management, Finance, HR, Marketing, Production, payroll, inventory processing	accounting (language of money, representations of accounts, reports), distribution (purchasing, supply chain management, distribution systems), finance, human resources (laws, compensation, recruiting, retention, training), marketing (the market, customers and customer satisfaction, market strategies, cycle time and product life cycle; environment scanning), production, international business
Business Problems and Appropriate Technical solutions, end-user solutions	business problems and appropriate technical solutions; quantitative analysis and statistical solutions; decision formulation and decision making; business intelligence systems; business use of spreadsheets, desk-top databases, presentation software, word processing and publishing
Modes of Business	B to B, B to C, C to C, B to G, C to G; organizational span (individual, work group, department, enterprise, inter-organization)
Regulations	Federal and State Regulations; compliance, audits, standards of operation (e.g. FAR); agencies and regulatory bodies
IT Standards	ITIL, CORBA
Business Law	legal system, courts, dispute resolution processes (mediation, arbitration, conciliation, negotiation, trial); types of organizations, contracts, warranties, and product liability; policy and management of intellectual property
Disaster Recovery	identify essential system functions to support business functions for restoration and recovery after a catastrophic failure; define requirements for critical system performance and continuity of business function; backup, replication, fail-over processes in support of system performance subsequent to a disaster
Enterprise Information Systems and Business Intelligence	Alignment of business processes with large system structures; configuration of large systems; implementation and training; integration with business intelligence capabilities and optimization of business procedure.

ISSN: 2167-1435
v30 n2572

IT Support for Business Functions	Business systems (budget, personnel, capital, equipment, planning, training, control); Specific systems (production, financial, accounting, marketing, supply chain, securities, taxation, regulation compliance)
Operational Analysis	scheduling, allocation, queuing, constraint theory, inventory management models, financial models, forecasting, real time analysis; linear programming, simulation
Managing the IS Function	Development, deployment, and project control; managing emerging technology; data administration; CIO functions; security management; disaster planning and business continuity planning
Information Center Service	PC Software training and support; application and report generators, IS Development, Development and operations staff; corporate application management, data safety and protection, disaster recovery

ISSN: 2167-1435
v30 n2572

5.0 Organizational Systems Development

Strategic Utilization of Information Technology	use of IT to support business process, integration of customer requirements; team development of systems, reengineering concepts and application, methodologies, interfaces, systems engineering, CRM and ERP concepts; Agile, Object, Lean UX and other methodologies; identification of security issues, incorporation of security concepts into designs ensuring security principles; development of IS policy
IT Planning	value of IT, integration of IT in reengineering, IT policy, end user advocacy and optimization, IT advocacy and alignment outsourcing / off-shoring (risks, benefits, opportunities), training; capture security controls and requirements, ensure integration of security objectives, assurance of people and information protection; ensure security in interface considerations
IT and Organizational Systems	types of systems relationship of business process and IT, user developed systems, use of packaged software, decision systems, social systems; information assurance and security designs; IT support of end-user computing, group process and computing, and enterprise solutions
Information Systems Analysis and Design	investigate, information analysis, group techniques / meetings design, systems engineering, Information architectures, enterprise IS development with strategic process; consideration of alternatives; application and security planning; conversion and testing, HIPAA, FERPA, ISACA, GAAP; requirements analysis. cost analysis, cost/benefit, satisfaction of user need / involvement, development time, adequacy of information assurance controls; consideration / adoption of emerging technology (e.g. mobile computing), consideration of optimal life-cycle methodologies and tools; physical design (database, interface design, reports design, programming, testing, system testing)
Decision Making	personal decision making, Simon's model, structured, unstructured decisions, decision tools, expert systems, advanced problem solving (Triz, Asit); business intelligence, advanced reporting technologies.
Systems Concepts, Use of IT, Customer Service	develop client relationships, understand and meet need, involving the client at all phases of the life-cycle; review of customer functional requirements; consideration of improved business process; assurance of customer needs into requirements analysis
Systems Theory and Quality Concepts	system components, relationships, flows, concepts and application of events and measurement, customer expectations, quality concepts; boundaries, open systems, closed systems, controlled systems; effectiveness, measuring system performance, efficiency
CMMI and Quality Models	quality culture, goals; developing written standards, templates; process metrics development process improvement through assessment, lessons

ISSN: 2167-1435
v30 n2572

	learned
Systems Engineering Techniques	scope development, requirements determination, system design, detailed design and specifications, Enterprise Architecture, System architecture, information architecture, make or buy, RFP/Bid Process verification and requirements tracing, validation planning and test case development, unit testing, integration, system testing, system certification, system acceptance, installation and operation of the system, post-implementation audit; ensuring security designs, secure configuration management; agency evaluation and validation of requirements; ensuring customer training and incorporation of installation teams
End-User Systems	individual software: word processing, spreadsheets, database, presentation, outlining, email clients, statistical packages; work-group software; enterprise software: functional support systems (e.g. PI), enterprise configuration
Enterprise Information Systems in Support of Business Functions	Systems that support multiple enterprise functions (e.g. SAP); Electronic Medical Record Systems for physician-groups, and for hospitals; Cloud solutions for individual and organizational support; TPS, DPS, MIS, EIS, Expert System
Emerging Technology	Bleeding edge technologies; testing and adoption of new technologies; cost benefit of new technologies
Systems Roles in Organizations	operations, tactical, strategic
Organizational Models	Hierarchical, Flow Models, Matrix
Metrics and Improvement	Development metrics, quality metrics, metrics in support of 6-Sigma or CMMI, customer satisfaction; Learning Cycles (Understand the problem, plan, act, measure/reflect and learn and repeat the cycle), Lessons Learned (what was supposed to happen, what happened, what was learned, what should be done, communicate the observations)
Hardware selection, acquisition, and installation for project	Determination of capacity for process, storage devices, and communication systems; consideration of alternative hardware; bid preparation, bid evaluation, and final system selection; hardware installation and testing; system deployment and initial operation.
Facilities Management	Physical facility construction, access control, fire protection, prevention of flooding; power management (public utilities, generators--fuel storage, testing, battery management--lightening protection), air conditioning, fire prevention systems, physical security, protection from weather

ISSN: 2167-1435
v30 n2572

Maintenance Programming	Fault detection and isolation, code correction, code testing, module testing, program testing; code, module, system documentation
Decision Structure	structured, unstructured decisions, decisions under uncertainty, heuristics, expert systems
Decision Tools	application results, idea generation, Delphi, nominal group, risk analysis, cost benefit analysis
Structured development	process flows, data flows, data stores, process logic, database design, program specifications and design
Object Oriented Development	UML; class diagrams, swim lane, use case, sequence diagram, design patterns
Screen Design	menus, input forms, output forms and reports, linkage of screen modules, navigation
Frameworks and Libraries	object libraries, source libraries, language extensions
Reports Development	simple lists, control break--group by--reports, error reports, exception reports, graphics reports, audit reports
Develop Audit Control Reports	Document new accounts with public information: names, addresses, organizations, items, events
Develop cash audits	deposits, batches, accounting variable controls, accounting distributions
Audit analysis of separation of function	establish roles of staff, validate transactions, validate personal functioning
Audit risk and disaster recovery strategies	determine risks, verify adequacy of mitigations; audit failure processes, replication, and failover mechanisms; audit backup strategy and physical results

ISSN: 2167-1435
v30 n2572

Table 6. Skills in Analysis and Design Course

Strategic Utilization of Information Technology	use of IT to support business process, integration of customer requirements; team development of systems, reengineering concepts and application, methodologies, interfaces, systems engineering, CRM and ERP concepts; Agile, Object, Lean UX and other methodologies; identification of security issues, incorporation of security concepts into designs ensuring security principles; development of IS policy
IT Planning	value of IT, integration of IT in reengineering, IT policy, end user advocacy and optimization, IT advocacy and alignment outsourcing / off-shoring (risks, benefits, opportunities), training; capture security controls and requirements, ensure integration of security objectives, assurance of people and information protection; ensure security in interface considerations
IT and Organizational Systems	types of systems relationship of business process and IT, user developed systems, use of packaged software, decision systems, social systems; information assurance and security designs; IT support of end-user computing, group process and computing, and enterprise solutions
Information Systems Analysis and Design	investigate, information analysis, group techniques / meetings design, systems engineering, Information architectures, enterprise IS development with strategic process; consideration of alternatives; application and security planning; conversion and testing, HIPAA, FERPA, ISACA, GAAP; requirements analysis. cost analysis, cost/benefit, satisfaction of user need / involvement, development time, adequacy of information assurance controls; consideration / adoption of emerging technology (e.g. mobile computing), consideration of optimal life-cycle methodologies and tools; physical design (database, interface design, reports design, programming, testing, system testing)
Decision Making	personal decision making, Simon's model, structured, unstructured decisions, decision tools, expert systems, advanced problem solving (Triz, Asit); business intelligence, advanced reporting technologies.
Systems Concepts, Use of IT, Customer Service	develop client relationships, understand and meet need, involving the client at all phases of the life-cycle; review of customer functional requirements; consideration of improved business process; assurance of customer needs into requirements analysis
Systems Theory and Quality Concepts	system components, relationships, flows, concepts and application of events and measurement, customer expectations, quality concepts; boundaries, open systems, closed systems, controlled systems; effectiveness, measuring system performance, efficiency
CMMI and Quality Models	quality culture, goals; developing written standards, templates; process metrics development process improvement through assessment, lessons learned
Systems Engineering Techniques	scope development, requirements determination, system design, detailed design and specifications, Enterprise Architecture, System architecture, information architecture, make or buy, RFP/Bid Process verification and requirements tracing, validation planning and test case development, unit testing, integration, system testing, system certification, system acceptance, installation and operation of the system, post-implementation audit; ensuring security designs, secure configuration management; agency evaluation and validation of requirements; ensuring customer training and incorporation of installation teams
End-User Systems	individual software: word processing, spreadsheets, database, presentation, outlining, email clients, statistical packages; work-group software; enterprise software: functional support systems (e.g. PI), enterprise configuration
Enterprise Information Systems in Support of Business	Systems that support multiple enterprise functions (e.g. SAP); Electronic Medical Record Systems for physician-groups, and for hospitals; Cloud solutions for individual and organizational support; TPS, DPS, MIS, EIS, Expert System

Functions	
Emerging Technology	Bleeding edge technologies; testing and adoption of new technologies; cost benefit of new technologies
Systems Roles in Organizations	operations, tactical, strategic
Organizational Models	Hierarchical, Flow Models, Matrix
Metrics and Improvement	Development metrics, quality metrics, metrics in support of 6-Sigma or CMMI, customer satisfaction; Learning Cycles (Understand the problem, plan, act, measure/reflect and learn and repeat the cycle), Lessons Learned (what was supposed to happen, what happened, what was learned, what should be done, communicate the observations)
Hardware selection, acquisition, and installation for project	Determination of capacity for process, storage devices, and communication systems; consideration of alternative hardware; bid preparation, bid evaluation, and final system selection; hardware installation and testing; system deployment and initial operation.
Facilities Management	Physical facility construction, access control, fire protection, prevention of flooding; power management (public utilities, generators--fuel storage, testing, battery management--lightening protection), air conditioning, fire prevention systems, physical security, protection from weather
Maintenance Programming	Fault detection and isolation, code correction, code testing, module testing, program testing; code, module, system documentation
Decision Structure	structured, unstructured decisions, decisions under uncertainty, heuristics, expert systems
Decision Tools	application results, idea generation, Delphi, nominal group, risk analysis, cost benefit analysis
Structured development	process flows, data flows, data stores, process logic, database design, program specifications and design
Object Oriented Development	UML; class diagrams, swim lane, use case, sequence diagram, design patterns
Screen Design	menus, input forms, output forms and reports, linkage of screen modules, navigation
Frameworks and Libraries	object libraries, source libraries, language extensions
Reports Development	simple lists, control break--group by--reports, error reports, exception reports, graphics reports, audit reports
Develop Audit Control Reports	Document new accounts with public information: names, addresses, organizations, items, events
Develop cash audits	deposits, batches, accounting variable controls, accounting distributions
Audit analysis of separation of function	establish roles of staff, validate transactions, validate personal functioning
Audit risk and disaster recovery strategies	determine risks, verify adequacy of mitigations; audit failure processes, replication, and failover mechanisms; audit backup strategy and physical results

Table 7. Data Management Topics in Course

Topic	*Description*
1. Data Definition	Data typing and relationship to information
2. Data Modeling	Implementing the requirements
3. Data Schema	Relate "real" objects to data representation
4. Entity-Relationship Diagrams	Specify the relationship between objects and how they may transform each other over time
5. Normalization	How to specify and structure schemas such that the observable relationships between entities can be maintained.
6. Referential Integrity	To ensure that we preserve the logical nature of relationships as dynamicity is introduced to the data store
7. Structured Query Language	The ability to issue instructions and questions to the RDBMS for results
8. Transactions	As the database lives and operates, data integrity is maintained by ensuring that transactions are *Atomic*, *Consistent*, *Isolated*, and *Durable*.
9. Concurrency Control	The RDBMS must handle multiple transactions and retain the ACID properties of each transaction.
10. Implementing the Data Model	Using DDL to write scripts to build the physical model
11. Ensuring Data Quality	Controlled attributes, data types

Table 8. Programming Topics in Courses

Topic	*Description*
1. Problem setting and problem solving	This is perhaps the primary competency that programming affords ANY student who undertakes the study of programming. Problem setting is an understanding of the problem domain and an articulation of the problem. Problem solving is the root of algorithmic thinking and the ability to harness the mental acuity that programming affords
2. Data (and Information)	Students must be aware of how data and information can be classified, quantified, notated, and accessed for transaction. Moreover, data transformed in act of problem-solving is often transformed as information.
3. Hardware and Software	Our students must be aware, even if at a cursory level – of the machines to which they communication and collude for computing outcomes.
4. Input and Output	If we accept that computer programs are reducible to Input -> Processing -> Output, then the basics of input and output, as facilitated by whichever programming language is being used, is covered
5. Logic	Students will solve problems, and transact and transform data, using tools that are rooted in logic. Logic is fundamental to problem solving as it prescribed valid reasoning
6. Algorithms	While both logic and algorithms can taken to a deep mathematical space, this will not be necessary for information systems students. Rather, algorithms teach students to express a methodical and reliable expression of their problem-solving.
7. Graphical User Interfaces	As a component of success is retention, our previous literature suggests that captivating and holding attention is key. Therefore, the event-driven Graphical User Interfaces that principally govern students' extant utilization of computing should appear earlier than later. The topic can't be mastered at this stage, but the immediacy and relevance of a GUI can help towards the journey of success.
8. Control Structures / Conditional Logic	This is the essence of programming and a stage where coaching, repetition, and similar antecedents to mastery must be engaged. These essential building blocks must be engaged throughout the period of instruction (at all levels leading up and during the capstone).
9. Debugging	Errors of syntax and semantics will plague both the beginner and professional. Tools, techniques, and strategies for reviewing and correcting these errors are fundamental.

ISSN: 2167-1435
v30 n2572

10. Data Structures (Static and Dynamic)	At the very least, both arrays (static) and linear/list (dynamic) data structures are requisite for dealing with data complexities of any "real world" project.
11. Modularity I (Functions/Subs/Methods)	As problems become more complex, students will need techniques and approaches for modularizing and compartmentalizing problems
12. Scope, Promotion, Casting	Now that problems and programs become more complex, handing data across and within modules becomes a concern.
13. Modularity II (OOP)	Object-oriented programming offers a paradigm such that programs can more closely fit the characteristics of the problem space. Makes programming "fit" more closely with data management and systems analysis and design.
14. Error Handling	Complex programs are executed in an equally complex computing environment such that anticipated failures can be addressed with error handling.
15. Data Persistence	The complex computing environment requires that data is persisted beyond runtime. This is where interaction with local and network stores becomes important.
16. Data Connectivity	Programs frequently "converse" with other programs over data communication networks. Students must be aware of how to programmatically utilize appropriate communication protocols (TCP, UDP, FTP, SSH, etc.)
17. Modularity III (Components, Libraries, etc.)	Once a complex set of programs are collaborating in unision, we are truly developing information systems. As this work is performed overtime, the need to both develop and utilize (internal and external) code and binary libraries becomes evident.
18. Data Structures II	Increase complexity also means that data may need to be stored and manipulated in more sophisticated data structures. Even if these data structures are appropriated by way of a vendor-supplied library of these structures (such as C++'s STL or Java/.NET's Collections API) , students will need to understand and use common data access and manipulation algorithms.
19. Generics/Templating	OOP and even data structures and algorithms benefit from generic and template designs.
20. Patterns and Principles	As students prepare to work with a systems development project, the phenomenon of patterns and principles are worth consideration. Decades of practice has wrought various meta, design, and architectural (MVC, etc.), "patterns" that codify "best practice.
21. Multi-threading and Parallel Computing	Programs will operate in a multi-tasking and multi-processing environment. Approaches to facilitating this in

ISSN: 2167-1435
v30 n2572

	programming languages are unique and must be studied (lest the students are fooled into thinking this is "magic").
22. GUI Variations (Web/Mobile)	While the web and mobile paradigms can be argued as proposing unique challenges which warrant consideration beyond "GUI variations," we use this term for convenience. Furthermore, library and tool support for developers on these platforms make workflows for making GUI-oriented programs very similar (Visual Studio, XCode, .NET and ASP.NET, Appcelerator Titanium, PhoneGap, Ximian, etc.)
23. Major Web-Oriented Application Development Framework (ASP.NET)	Ultimately, to write comprehensive and contemporary information systems/software solutions, students will need to be versed in a technology stack that provides full-service features for systems development – design, development, libraries, frameworks, testing, implementation, modification, and update. A representative tool for this is Visual Studio and the .NET Framework. For instance, in the web space, ASP.NET and ASP.NET MVC can integrate with Visual Studio, .NET, IIS, and Microsoft SQL Server for a fully-integrated development experience.

ISSN: 2167-1435
v30 n2572

Table 9. Possible Failure Mechanisms in Programming Courses

<table>
<tr><td>Lack of Student Maturity</td><td>There is little doubt that students exhibit a paucity of maturity in many regards - failure to come to class; failure to have an attitude of success; immature reaction to our correction; and returning a haughty response.</td></tr>
<tr><td>Lack of Preparedness from High School</td><td>Some of the difficulty in achieving success may be due to high school preparedness (Shannon et al 2012) in critical thinking (Shannon et al, 2009, 2010). In those studies, almost 90% of high school students could not apply critical thinking skills. Basically, students entering into college-level work are not capable of interpreting data, of problem solving, or of proposing solutions.</td></tr>
<tr><td>Failures of Delivery</td><td>Lecturing to the student about computer programming is often not a successful method to teach such topics. Students should have hand-on sessions in a computer lab setting to “try” specific programming tasks along with the instructor. A lack of hands-on work with the instructor can lead to frustration on the student’s part when he runs into various “strange” computer behavior, such as integer math operations.

A student may come to the course unprepared in terms of a background in logical thinking and or mathematics. They could also have a fear of a programming course because they have a fear of math. A good instructor can create good exercises in both logic and math operations to help encourage the student lacking in these skills.</td></tr>
<tr><td>Moving too fast</td><td>It is always difficult to decide what the pace of the course should be. There will be some students who will understand the programming concepts better than others. There will also be some who will never understand the concepts. This is the age old question: “what should the pace of the class be”? Perhaps at the beginning of the class meeting, the instructor could announce what concepts are being covered on that particular day. The students who report they have read the section in the book and are comfortable with the topic could be sent to a lab on campus to complete an exercise with that concept. Thus they can work at a faster pace than those who stay to listen to the entire lecture on the concept. One of the authors has successfully used this method with several students each semester. Another related issue is the lack of available tutoring for advanced students. Often there are tutors available for a beginning programming course, but not for an advanced one. Therefore, a great deal of pressure falls on upon the instructor to provide help outside of class to struggling students. Perhaps a cohort or group-based project could help with this issue.</td></tr>
<tr><td>Failures of Curricular Structure</td><td>There are a number of factors which might contribute to failure in the introductory programming sequence (Hoskey, et al, 2010). Worse yet, “Numerous studies document high drop-out and failure rates for students in computer programming classes. Studies show that even when some students pass programming classes, they still do not know how to program.” (Hoskey et al 2010):

<table>
<tr><td>Time Lapse Since Programming 2</td><td>Yes</td></tr>
<tr><td>Introductory Programming Language</td><td>Yes</td></tr>
<tr><td>Track (Concentration)</td><td>Yes</td></tr>
<tr><td>General GPA</td><td>Yes</td></tr>
<tr><td>Logic Course</td><td>No</td></tr>
<tr><td>Major</td><td>No</td></tr>
<tr><td>Faculty</td><td>No</td></tr>
<tr><td>Gender</td><td>No</td></tr>
<tr><td>Number of Programming Courses Taken</td><td>No</td></tr>
<tr><td>Math Courses Taken</td><td>No</td></tr>
</table>
Table from Hoskey et al, 2010 showing variables explored and significance in failure from programming class.</td></tr>
</table>

ISSN: 2167-1435
v30 n2572

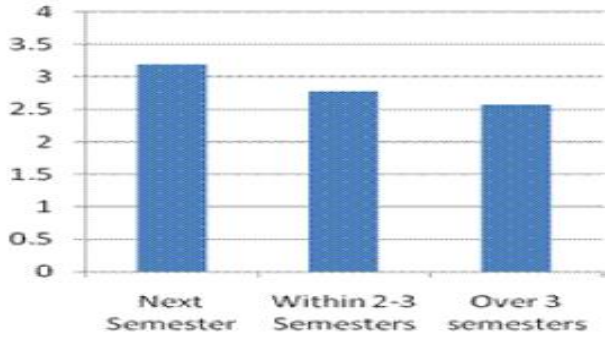
Delay Between Courses	 Figure from Hoskey et al, 2010 shows a clear decrease in student performance as time is allowed to pass after taking the initial class.
Failures of Leadership, Motivation, and Correction	As is the case with John Wooden's advice on coaching, our student's flaws are not entirely our fault or responsibility, but they are our problem. While this position may be anathematic to many, a coaching pedagogy would dictate that we engage all of our students with leadership, maturity, and correction.
Choice of Wrong Programming Language	White (2012) studied the impact on cognition while learning Visual Basic. He found that "… since left hemispheric cognitive style is required to be successful in Visual Basic and Visual Basic does not create such cognitive style, this research, as well as other research, supports the need for prerequisites for Visual Basic to ensure students' success."
More with Less	A common mistake, given the length and depth of topics we must teach in the programming sequences, is to attempt to cover the "full menu" of topics. In the interest of positive forward progress and the maintenance of confidence and focus, students can gain the essence of a topic without full grasp. For instance, for an information systems student, can the while-loop be taught as their only tool for repetition structures? For a considerable length of time, this tool will suffice. It may turn out then, that after using the while-loop and developing real craft and mastery with it, the introduction of the for-loop may be trivial and intuitive to the student with a greatly reduced (if not eliminated) period of instruction from the educator.
Scope Creep versus Establishing Confidence	We seek to avoid scope creep (or the "kitchen sink" approach) as our aim is success – we can't teach them everything. Rather, we advocate for an approach which lets students be good now! We advocate for confidence over competence. We must be very careful to avoid over complicating presentations lest students fall into a "whooped dog syndrome".
Trying to Put too Much in the Course	We are reminded of this maxim: it takes time to build humans. Do not go faster with the material than the class can handle. As new programming tools are introduced, students should be given assignments and/or lab work with those tools. They need time to incorporate the new programming tools into their own programming knowledge base. The makeup of the students in a programming course will differ each time the course is taught, therefore, what is covered may be slightly different from one semester to another. It will not be possible to cover everything in the text book. The instructor should develop a list of "must have" topics that are required teaching each semester and another list of "add on" topics that could be taught if the class is progressing at a faster pace.

Table 10. Successful Approach in Teaching Programming

Everyone Can Succeed	The character in the Movie "Stand and Deliver" portrays is a real-world Jaime Escelente high school Advanced Placement (AP) Calculus teacher. What is remarkable about his story is that ALL of his students PASS the difficult AP exam. What is even more remarkable was that all of the students were fraught with many life-problems. His devotion and coaching style netted 17 years of repeat performance. "He rejected the usual markers of academic excellence and insisted that regardless of a student's GPA, he would let her take the AP course if she promised to work hard." (Mathews, 2010)
Use Tutoring	When especially difficult topics are introduced, such as methods/functions and abstraction through class definition, extra class sessions could help. Tutoring on campus is also something that is extremely beneficial. Good student should be encouraged to sign up to be tutors. We should use these advanced students as experts. Interestingly, it is a wonderful way for the student who is doing the tutoring to increase their own coding skill set. Debugging someone else's code is difficult and helps both students on each side of the tutoring process.
Write Short Sample Programs	Writing very short sample programs to illustrate one specific programming concept can be very helpful to the student. Often the programming books are difficult for the student to read, in that a specific programming concept may be embedded within a complicated example. The student is just trying to see where to put the beginning ending braces for a loop and the book may have an example of a loop running across three pages. Therefore, the sample code can be a valuable resource to the students.
Project Based Learning	Researchers have investigated project-based learning in a wide variety of disciplines and settings. They have generally found project-orientation to be effective in increasing student motivation, improving student problem solving, improving higher-order thinking skills, addressing different learning styles, and providing students with an integrated learning situation (Hutchings & Wutzdorff, 1998). The successful completion of each programming assignment should also include the analysis and design of the problem, data requirements and logic needed to code and test the program. Students should be required to turn in their pseudo code and/or their design with each programming coding effort. Programming can be viewed as a process of building a plan, in the form of source code, to achieve a certain goal (McCauley, Fitzgerald, Lewandowski, Murphy, Simon, Thomas, and Zander, 2008). But Kazemian and Howles found that among the students they surveyed, only 5 percent of the students always developed a design prior to starting to program (Kazemian, Howles, 2008). Therefore, designing a solution before coding should be a requirement for the first time programmer.
Large Coding Projects	In one study it was found that the beginning programmer can create very large projects by focusing on the programming concepts as they are needed for the project (Baugh and Kovach, 2012). In the course studied, students were first given a number of small programs to write that highlighted the basic building blocks of programming; input, output, variables, math operations, selection (if) statements, loops, and other control structures. Then a programming project was initiated that required these concepts, as well as the use of new programming concepts, as each phase of the project was assigned. This method of teaching programming showed a great deal of success with 60% of the students receiving an A for the course. Semester coding projects allow the student to see the real value of a computer program and what may be required in the real world. The students also feel a great sense of pride and accomplishment in the creation of a programming project that is many, many lines and pages of code. Susan Wojcicki Senior Vice President, Google said "Learning to code makes kids feel empowered, creative, and confident." (http://www.code.org/quotes) Students who are empowered, creative, and confident are the qualities we expect to witness in those who have successfully

	completed a large programming project.
Cohort learning	Creating a cohort of students to go through the curriculum together is another approach that has shown success in other areas of IS education. In one study, IS Doctoral students who worked in a cohort for the three years of their study reported that the main reason they felt they were successful was because of the cohort approach (Baugh, Kohun, 2005). The students reported that they were in it "together" and would do whatever they could to insure that all "made it". With a cohort approach, multiple programming courses with continuity from one course to the next would be easily accomplished. Students would be cheerleaders for each other.
Group/Team Projects	Another possible method of teaching programming that may be successful is allowing students to work in groups or teams on a coding project. This approach is often used on lab assignments by these authors. Students working together and helping each other can definitely have great benefits. But for writing code, care must be taken to ensure that everyone is helping with the code and it is not just one person who is taking on most of the work. A way to ensure this is to test the students on all concepts that are required in the specific programming assignment. One instructor has all students explain their semester project code on a final to insure that they actually wrote the code (Baugh, 2009). The student will not be able to adequately explain the code if they did not write it or help to write it.
Make it Relevant to the Student	Making the course material relevant to the student has shown to produce both increased student interest and success (Baugh, 2011). If the course material can be made more interesting to the student, then he will be more inclined to learn it. A real world project allows the students to "learn better through a particular domain of their interest" and "see the practical value of what they learned." (Robbert, 2000) Should an Introductory programming course be taught differently than an doctoral level course? The first answer that one might give to this question is yes, of course. But although the course work is obviously different, the same approach for assignments can be used in almost any IS course. If the course material can be made more interesting to the student, then he will be more inclined to learn it. A real-world project allows the students to "learn better through a particular domain of their interest" and "see the practical value of what they learned" (Robbert, 2000). One instructor designed a way to teach beginning C++ where the students choose their own area of interest, and thus created their own data set. (Baugh, 2011) Students wrote a menu-driven program that was broken up into phases. At the completion of the project, each student's project performed the following tasks utilizing the data of the students choice: • Read data from data files • Wrote data to data files • Implemented various class structures • Manipulated data in multi-dimensional arrays, including inserting, deleting and modifying • Coded various error checking functions • Coded various search functions • Coded reports • Wrote user's and programmer's guides for the project At the completion of this course, students reported that they were very proud of the large coding project they had written. Again, most of them had no previous programming experience. A number of the students said that they spent a great deal of time on the project, but because of the individualized data, the extra time was something they did not mind. They reported that they felt that the project was more interesting to work on because the data was of interest to them. One student said "without a doubt this was one of the best classes I have ever taken." Even the beginning programmer can write a large project.

ISSN: 2167-1435
v30 n2572

Bring in Former Students	Bringing in former students from recent grads to accomplished professionals gives students the opportunity to see how they might be transformed in a few short years. Such speakers might spend time talking about how they made this transition—yes, it took a lot of hard work, but the benefits have become so large and exciting such as the ability to support a family comfortably…!