A Constructivist Approach to Teaching Web Development in Post-Secondary Vocational Settings

John M. Bunch
The SAJES Group, Inc., Tampa, Florida, USA

jmbunch@sajes.com

Executive Summary

Vocational education by its nature has a need for delivery methods that place a strong focus on the relationship between school and work and seeks to deliver instruction in a manner that bridges the two as seamlessly as possible. This paper presents a curriculum and constructivist-based instructional delivery approach, designed to emphasize a strong school-work relationship, for a four-year degree in Web Development developed for a vocational training institution. The instructional outcomes for such a program are distinctly different from traditional software engineering and other computer -specific programs and require a different approach to curriculum development and instructional delivery, which focuses on the unique needs of vocational students. At the same time, such programs should strive to emulate the best practices, educational values, and, to the extent possible, the curriculum of traditional programs. The educational program presented here employs a spiral sequencing of course material, presented using the constructivist approach of goal-based scenarios, in order to emphasize the applied, skill-building nature of vocational instruction. Many authors have discussed the benefits of a constructivist approach to vocational education (i.e. Brown, 1998), while others have called for its increased use in computer science related education (i.e. Connolly & Begg, 2006).

The current program adheres as closely as possible, given its vocational mission, to the latest recommendations and guidelines concerning four-year degree programs in software engineering from the ACM/IEEE Joint Task Force on Computing Curricula.

Keywords: Problem-based learning, constructivism, career and technical education, post-secondary education, Web development

Introduction

This paper presents a curriculum and instructional delivery approach developed for a "bachelors" program in Web application development at a post-secondary career education / technical training school. While a technical subject to be sure, Web Development brings together skills sets from a variety of business, computer science, and art related disciplines to produce (as noted by

Editor: Keith Willoughby

Deshpande & Hansen, 2001) a “discipline among disciplines” with its own unique goal (i.e. to produce Web applications), synthesized knowledge base, and emerging standards and practices.

As a new field, it lacks universally implemented post-secondary curricula with the specific goal of producing professional Web developers or engineers. Currently, Web professionals typically have formal post-secondary training in one related discipline, such as network engineering, database design, or multimedia design. As noted by Whitehead (2002), the primary skills of a Web professional are an integration of several disciplines, and the demand is growing for educational programs focused on delivering a holistic curriculum specific to that integration.

To that end, the current program seeks to create a coherent curriculum for Web application developers and, at the same time, a delivery strategy focused on the needs of post-secondary career/technical education (CTE) students. The first part of this paper outlines the process of defining the current curriculum within the context of existing college-level curricula in related fields. The second part discusses the constructivist approach developed to implement it with post-secondary CTE students.

Locating Web Development within the Joint Task Force Computing Disciplines

Program learning objectives and the corresponding specific curriculum areas were set by an advisory board of industry professionals and area employers in cooperation with faculty (all of whom had professional experience as Web developers). Four general curriculum content areas were identified, corresponding to the typical tier-based architectural conceptualization of Web applications (user interface, business logic, and data services), along with global areas applying to all tiers (e.g., project management, understanding business goals, computer networking, professional practice, and ethics). ASP.Net using Visual Basic.Net was selected as the business logic programming environment.

The bachelor's curriculum presented here follows an associate's program in graphic design that includes an introduction to bitmapped graphics and Web page creation using graphical tools such as Macromedia's Dreamweaver, but provides no instruction whatsoever in programming or scripting or the other topics presented in the bachelor's program. Also, it requires no math or science prerequisites. In addition to liberal arts courses, the bachelor's program consists of eight four-credit hour courses and one internship, all taken over six quarters.

While the setting of overall program learning objectives for the curriculum presented here was accomplished by a fairly straightforward process, it was important to situate these objectives within an overall context of computing knowledge. In other words, it was not our goal to define a Web Development curriculum orthogonal to a traditional curriculum in computer science, information systems, and other computer-related programs. Rather, it was to understand where the current curriculum fit within the context of those programs and to integrate the best practices and historical lessons of educational programs in those areas wherever possible.

The *Computing Curricula 2005: The Overview Report* (ACM/IEEE, 2006) discusses five computing disciplines for which specific curriculum guidelines and recommendations have been published: Computer Engineering, Computer Science, Software Engineering, Information Systems, and Information Technology. These five emerged during the 1990's from the more well-defined fields of Electrical Engineering, Computer Science, and Information Systems as new developments and increasingly broad use of computational devices grew. While there is certainly overlap with each of the five areas and *some version* of a Web Development curriculum could reasonably be generated from within any of them, each with its own emphasis, the learning objectives for the curriculum presented here locate Web Development within the discipline of Software Engineer-

ing. It is not the aim of this paper to define the subject matter content for a universally acceptable curriculum for "Web Development," but rather to discuss the implementation of one specific example.

Both Web Development, as we have defined it within this curriculum, and Software Engineering, as defined by ACM/IEEE, share the description of "...developing and maintaining software systems that behave reliably and efficiently, are affordable to develop and maintain, and satisfy all the requirements that customers have defined for them (ACM/IEEE, 2006, p. 15)." In contrast, Computer Engineering balances an emphasis on software with an emphasis on hardware. Computer Science places an emphasis on greater theoretical breadth. Information Systems address the solution of a broad scope of organizational information needs using a wide range of computer technology, and the newer discipline of Information Technology is a complement to Information Systems, in that "the emphasis is on the technology itself more than on the information it conveys" (ACM/IEEE, 2006, p. 14).

Also, when considering expectations of program graduates, Software Engineering is unique among the five areas in that its focus is strictly on competencies in software development, as opposed to organizational computing infrastructure (Information Technology), designing systems to meet general organizational and business goals (Information Systems), competency with theoretical work (Computer Science), and the integration of hardware and software (Computer Engineering).

While the subject matter of the current curriculum is discussed in greater detail elsewhere in this paper, Table 1 presents the deviations of the comparative weights of the knowledge areas of the current program from the minimum and maximum weights of those of the five kinds of degree programs as defined in ACM/IEEE (2006). To obtain the deviation scores, the comparative weight of each knowledge area of the current program was subtracted from the minimum and maximum weights for each knowledge area across the five kinds of degree programs (from Table 3.1, ACM/IEEE, 2006). As with ACM/IEEE (2006), comparative weights are assigned as a rating, from zero to five, representing the relative emphasis on each knowledge area.

The number of individual knowledge areas in which the comparative weight of that knowledge area within the current program was congruent with (equal to or greater than the minimum, and equal to or less than the maximum) the same knowledge area within each of the five types of degree programs were tallied. Of the 40 knowledge areas, the current program was congruent with the comparative weights of 32 within Software Engineering degree programs, 17 within Information Technology, 23 within Information Systems, 23 within Computer Science, and 19 within Computer Engineering. Thus, Software Engineering, with its greater emphasis on assessing customer needs and developing "...usable software that meets those needs (ACM/IEEE, 2006, p. 15)," is the parent discipline most closely encompassing the learning objectives and career goals defined for the Web Development program presented here.

The Web Development curriculum goals defined here are clearly much narrower in scope than those defined for software engineering within ACM/IEEE (2004), thus the program presented here is perhaps best thought of as a sub-area or specialty within software engineering that does not seek to impart the breadth of related domain knowledge. The priority is on the acquisition of applied skill rather than breadth of conceptual knowledge, although as much effort toward the latter is made as can be accommodated while maintaining a focus on the former.

Table 1. *Deviations* of comparative weight of computing topics across the five kinds of degree programs with the current Web Development curriculum*

Knowledge Area	Current Program	Deviation Scores									
		CE		CS		IS		IT		SE	
	(weight)	min	max	min	max	min	max	min	max	min	max
Programming Fundamentals	5	1	1	1	0	3	1	3	1	0	0
Integrative Programming	2	2	0	1	-1	0	-2	-1	-3	1	-1
Algorithms and Complexity	3	1	-1	-1	-2	2	1	2	1	0	-1
Computer Architecture and Organization	2	-3	-3	0	-2	1	0	1	0	0	-2
Operating Systems Principles & Design	2	0	-3	-1	-3	1	1	1	0	-1	-2
Operating Systems Configuration & Use	2	0	-1	0	-2	0	-1	-1	-3	0	-2
Net Centric Principles and Design	3	2	0	1	-1	2	0	0	-1	1	-1
Net Centric Use and configuration	3	2	1	1	0	1	-1	-1	-2	1	0
Platform technologies	2	2	1	2	0	1	-1	0	-2	2	-1
Theory of Programming Languages	1	0	-1	-2	-4	1	0	1	0	-1	-3
Human-Computer Interaction	4	2	-1	2	0	2	-1	0	-1	1	-1
Graphics and Visualization	4	3	1	3	-1	3	3	4	3	3	1
Intelligent Systems (AI)	0	-1	-3	-2	-5	-1	-1	0	0	0	0
Information Management (DB) Theory	2	1	-1	0	-3	1	-1	1	1	0	-3
Information Management (DB) Practice	4	3	2	3	0	0	-1	1	0	3	0
Scientific computing (Numerical mthds)	0	0	-2	0	-5	0	0	0	0	0	0
Legal / Professional / Ethics / Society	3	1	-2	1	-1	1	-2	1	-1	1	-2
Information Systems Development	2	2	0	2	0	-3	-3	1	-1	0	-2
Analysis of Business Requirements	5	5	4	5	4	0	0	4	3	4	2
E-business	4	4	4	4	4	0	-1	3	2	4	1
Analysis of Technical Requirements	5	3	0	3	1	3	1	2	0	2	0
Engineering Foundations for SW	1	0	-1	0	-1	0	0	1	1	-1	-4
Engineering Economics for SW	2	1	-1	2	1	1	0	2	1	0	-1
Software Modeling and Analysis	4	3	1	2	1	1	1	3	1	0	-1
Software Design	5	3	1	2	0	4	2	4	3	0	0
Software Verification and Validation	4	3	1	3	2	3	2	3	2	0	-1
Software Evolution (maintenance)	4	3	1	3	3	3	2	3	2	2	0
Software Process	4	3	3	3	2	3	2	3	3	2	-1
Software Quality	4	3	2	3	2	3	2	3	2	2	0
Comp Systems Engineering	2	-3	-3	1	0	2	2	2	2	0	-1
Digital logic	0	-5	-5	-2	-3	-1	-1	-1	-1	0	-3
Embedded Systems	0	-2	-5	0	-3	0	0	0	-1	0	-4
Distributed Systems	4	1	-1	3	1	2	0	3	1	2	0
Security: issues and principles	3	1	0	2	-1	1	0	2	0	2	0
Security: implementation and mgt	3	2	1	2	0	2	0	0	-2	2	0
Systems administration	1	0	-1	0	0	0	-2	-2	-4	0	-1
Management of Info Systems Org.	0	0	0	0	0	-3	-5	0	0	0	0
Systems integration	1	0	-3	0	-1	0	-3	-3	-4	0	-3
Digital media development	3	3	1	3	2	2	1	0	-2	3	2

*Deviation scores were derived by subtracting the topic area weight of the current program from the minimum and maximum topic area weights for each of the five kinds of degree programs listed in Table 3.1 of ACM/IEEE (2006).

Traditional Academic Approaches to Teaching Software Engineering

Generally speaking, computer science curricula are delivered using a *topical sequencing* approach. As indicated by Reigeluth (1999), in topical sequencing a specific topic is taught to the required depth of understanding before moving on to the next. The ACM/IEEE (2004) presents this form of sequencing in its model curricula, and the teaching of programming (see Robins, Rountree, and Rountree, 2003 for a review) and database design (Connolly & Begg, 2006) are traditionally approached from this perspective.

Common prescriptions for computer programming curricula support this framework. The ACM/IEEE's recommended software engineering curriculum is organized into specific *Software Engineering Education Knowledge* (SEEK) areas. ACM/IEEE (2004) lists nineteen guidelines for the design and delivery of curricula intended to teach the SEEKs. Some of them, while certainly critical, are not particularly salient in the context of this discussion (such as that courses and curricula should be regularly reviewed and updated). Other guidelines, however, bear directly on the nature of the curriculum and the teaching process. These can be summarized as follows:

- Curriculum designers and instructors should be focused on *outcomes or learning objectives*.
- A software engineering mindset be developed through *recurring themes* in the curriculum.
- Students should be instilled with the *ability and eagerness to learn*.

- Software engineering should be taught as a *problem-solving* discipline.
- *Underlying principles* should be taught rather than the details of specific tools, although appropriate and up-to-date tools should be used.
- The curriculum should have a *significant real-world basis* and should incorporate project-based classes, practical exercises, and work experience.
- *Concrete and convincing examples* should be used to *motivate* students.
- A *variety* of teaching and learning approaches should be used.
- Important efficiencies and synergies can be achieved by designing curricula so that *several types of knowledge are learned at the same time*.

ACM/IEEE (2004) also includes an example course sequence which suggests how the SEEKs can be divided into courses then sequenced to produce the prototypical four-year degree program, some close facsimile of which is found at the majority of colleges and universities around the world. This curriculum is characterized by grouping the SEEKs into more or less discrete courses and ordering the courses into parallel linear sequences as appropriate, such as Programming Fundamentals --> The Object-Oriented Paradigm --> Software Construction --> Software Design and Architecture, etc., and culminating in a project-based capstone course designed to pull all of the SEEKS together and integrate them into a whole.

The guidelines for a two-year degree program in Computer Science follow a similar pattern (ACM/IEEE 2002). In this case, for example, the Programming Fundamentals course is followed by The Object-Oriented Paradigm, which is followed by Data Structures and Algorithms in the third semester. Four or five elective courses (such as Web-Centric Computing), about half of which have no prerequisites, complete the degree program. As with the four-year program, the two year program is designed for students who wish to enter the workforce upon degree completion *or* enter a higher-level degree program. As such, the goals of both the two-year and four-year degree programs recommended by the Joint Task Force are by definition broader than the goals of a vocational training degree program – the vocational training program is willing to sacrifice breadth and depth of related subject matter knowledge for practical skill within a narrower area, as the preparation for further formal education is beyond the scope of the program.

Understanding the Needs of Post-Secondary CTE Students

Brown (1998), in her review of the application of constructivist principles to vocational education, identifies a specific advantage of the current approach for vocational training when she states that "Vocational educators have long recognized the importance of connecting school to work (p. 14)." Further, research suggests that students in career and technical education programs prefer hands-on, engaged learning experiences (Ausburn & Brown, 2006; Gentry, Peters, & Mann, 2007; Stitt-Gohdes, 2001) and want to know that the skills they are being asked to learn will directly benefit them in the job market (Gentry, Rizza, Peters, & Hu, 2005). Anecdotally at least, this was by far the most commonly voiced opinion of the students currently enrolled at the institution for which the current program was developed.

These students expect to leave the post-secondary educational program with ready-to-use job skills, and the perception that they are acquiring those skills throughout their efforts in such programs works is an important motivational driver. However, for such programs to be effective beyond the short term and prepare the student for programming endeavors beyond the relatively brief nature of the immediate post-training context, the curriculum designer must seek to build the *schemas* and *mental models* for post-training programming skill acquisition and expertise to develop efficiently. As noted by Robins, Rountree, and Rountree (2003), a *schema* is the basic cog-

nitive chunk or plan used in program design and understanding, and *mental models* are mental representations of things such as problem domain or domains for which a program is to be written, the machine on which a program will run, and the completed program itself.

In their review of the literature on learning and teaching computer programming, Robins, Rountree, and Rountree (2003) propose a "programming framework" to be considered by educators when designing programming course curricula. This framework identifies three components of schema to be constructed within the mind of the student - knowledge, strategies, and models – for three primary task areas or stages within program development – design, generation, and evaluation. The knowledge component, for example, would involve planning and formal methods at the design stage, language syntax and libraries at the generation stage, and debugging tools at the evaluation stage. The ACM's SEEK areas can relatively easily be sorted into knowledge, strategies, and model schema components.

Thus our goal was two-fold. First, to create a Web development curriculum that, while much more narrow in scope than a traditional software engineering program, followed industry standard curricula and educational guidelines to the maximum extent possible given the CTE context. Second, to deliver that curriculum in a manner that would leverage the learning preferences and expectations of this audience of learners, while being mindful of the cognitive processes (a la Robbins et al., 2003) necessary to develop legitimate understanding and skill.

Instructional Method

As mentioned above, ACM/IEEE (2004) suggests a topical sequencing approach to instructional delivery. The current program, however, implements *spiral* sequencing (Bruner, 1960; see also Harden & Stamper, 1999), in which the learner masters a part of one topic before moving on to the next and mastering part of it, then the next, etc., before coming back to the first topic and mastering another part, and so on. Spiral content sequencing allows for the creation of a series of increasingly complex Web development tasks, each one implementing an ever larger piece of each application tier (i.e., user interface / business logic / data services). As noted by Collura, Aliane, Daniels, and Nocito-Gobel (2004) in their discussion of the recent development of a spiral approach to multidisciplinary engineering education at the University of New Haven, there is a relative paucity of attempts at implementing a spiral model at the majority of schools of engineering.

However, DiBasio, Clark, Dixon, Comparini, and O'Connor (1999) report that a spiral curriculum in chemical engineering boosted subject matter interest and communication skills, while Trumper (1996) notes that such an approach to teach basic concepts of energy would avoid many of the pitfalls inherent in modularized instruction and be more consistent with the contemporary perspective of constructivist learning. Cornford (1997), in a review of the educational psychology literature related to knowledge and skill acquisition in the context of vocational training, suggests that a spiral curriculum approach is much more consistent with the psychological processes of schema construction and mental model building than a traditional, modularized approach.

In the current program the spiral content sequence is delivered through the extensive use of *goal-based scenarios* (GBS) (i.e., Schank, Berman, & MacPherson, 1999). Each course is composed of a sequence of business scenarios which the student solves (usually) by developing a Web application. These scenarios progress from the simple to the complex, each requiring a slightly wider range and greater depth of knowledge and skill (sampled from each of the major topic areas) than the previous. In this way the student can see explicit, positive progress toward career skills throughout each individual course.

While such an approach is unusual within software engineering education, many educators have called for wider implementation of it. As noted by Merrill (2002) in his synthesis of the past several years of instructional design theories and models, the engagement of learners in real-world problems promotes learning. Merrill defines a real-world *problem* as one in which the learning activity represents a whole task, and the task is representative of problems the learner would encounter in the real world. As noted by Brown (1998), problem-based instruction reflects a constructivist approach to teaching. Brown's (1998) comprehensive report on the application of constructivist practice in vocational and career education finds that constructivism is consistent with research findings on the neurological basis of cognition, seeks a high standard of intellectual quality, and connotes a new paradigm in teaching.

Bennett, Grasel, Parchman, and Waddington (2005) note that an increasing trend within science education is the desire by students to see instruction grounded within a real-world context, and that such instruction provides enhanced motivation for learners. Connolly and Stansfield (2006) note that information systems problems can be characterized by incomplete, contradictory, and changing requirements, and solutions can be difficult to recognize because of these complex interdependencies. Connolly and Stansfield (2006) further suggest that embedding learning activities in real-world problems can help overcome these issues. Connolly and Begg (2006) argue that problem-based instruction can be particularly helpful in teaching the abstract concepts inherent in database design, one of the application tiers Web developers must master. Connolly and Begg (2006) describe the difficulties in effectively teaching design concepts in the traditional format and suggest that the learning-by-doing model of problem-centered instruction is needed to effectively impart these skills. Linder, Abbott, and Fromberger (2006) make a similar argument in the arena of software design.

Cartelli, Stansfield, Connolly, Jimoyiannis, Magalhaes, and Maillet (2008) argue for constructivist, problem-based learning in online learning for information technology education, while Leitch and Warren (2007) argue for problem-based learning to teach information systems in general in Australian universities. Cheong (2007) reports on the successful use of problem-based learning to teach an intelligent systems course and notes the importance of structure in a problem-based implementation. Venebles and Tan (2007) successfully used a hands-on, problem-based approach to teach genetic algorithms to undergraduates.

Thus, the research literature provides a reasonable expectation of its instructional efficacy for the subject matter being taught. In addition, our instructional delivery method seems uniquely suited to vocational education - instruction is firmly grounded in a real-world context using real-world tasks, and the clear relationship between each instructional task and job skills is consistently (and constantly) maintained.

As noted by van Merrienboer, Kirshner, and Kester (2003), a potential problem with problem-based instructional approaches is that the learner becomes overwhelmed by task complexity, causing a level of cognitive load that interferes with learning. As discussed by Tuovinen and Sweller (1999), high cognitive load occurs when the many elements of instructional material compete at once for cognitive resources – primarily those of working memory.

An instructional approach useful for creating appropriate levels of cognitive load is *scaffolding* – presenting performance supports as needed to achieve a goal, then fading them away as the learner is able to achieve the goal on his or her own (van Merrienboer et al., 2003). The current program makes use of *worked-out examples*, in which cognitive load is reduced by having learners complete small parts of worked-out examples, then progressively complete larger parts of additional worked-out examples until the target task is accomplished (Tuovinen & Sweller, 1999).

Many authors have noted positive results in the use of worked-out examples for the acquisition of complex cognitive skills across a variety of domains (i.e. Hilbert, Renkl, Schworm, Kessler, &

Reiss, 2008; Yaman, Nerdel, & Bayrhuber, 2008; see Pass & van Gog, 2006, for a review), and note that the modularization of problem solutions into smaller solution elements itself works to reduce cognitive load (Gerjets, Scheiter, and Catrambone, 2004).

By implementing a spiral curriculum using goal-based scenarios, the program presented here attempts to follow the Joint Task Force Software Engineering curriculum design and delivery guidelines, along with the Robins, Rountree, and Rountree's (2003) framework for teaching computer programming. It also attempts to include the SEEKs defined for undergraduate degree programs in software engineering (ACM/IEEE, 2004). Admittedly, however, the breadth of content within the SEEKs is much narrower in the current program than would be typical of a traditional software engineering degree program. The intention is to provide an example or suggestion for other educators faced with a similar task – providing legitimate, quality instruction with the goal of developing hands-on, applied, real-world knowledge and skills.

Curriculum Development and Content Sequencing

A three-semester track of *tier practice* courses, titled *Introduction to Server Side Programming*, *Intermediate Server-Side Programming*, and *Advanced Server-Side Programming*, are taken during quarters 1, 2, and 3. These courses are each composed of a series of goal-based scenarios designed to require an ever increasing expertise at each level of the Web application tier.

For example, in a sample lesson from Introduction to Server Side Programming, students are given the following GBS:

> *You have been asked to develop a Web site for a children's baseball league. Along with a description of the league, other marketing text, and pictures, the league would like to have a "contact us" form that site visitors can use to submit their name and contact information. The league wants to be able to go to a special administration page and see a list of all information collected.*

A solution to this scenario is then demonstrated by the instructor, with knowledge and skills presented from each content area (thinking through and planning a solution, creating HTML documents, creating ASP.Net files to process HTML form data and retrieve data from the database, creating a database table to store the data). Within each content area specific content items are explained and demonstrated (HTML form tags, the Response.write method, ADO.Net connection object, Datagrid control, SQL Insert statement, etc.).

Once the GBS has been thoroughly demonstrated by the instructor, the students are given the solution as a worked-out example. Other materials and scaffolding techniques are used as appropriate, generally at the determination of the instructor. Students are then given the following GBS to complete independently:

> *You have been asked to develop a Web site for a local antique automobile collector's club. In addition to general information about the club, pictures of antique automobiles, etc., the club would like to have a page where antique automobile owners can register for an upcoming auto show. The club wants to collect the registrant's name and contact information, along with the make and model of the automobile they will bring to the show. Finally, the club wants another page that displays a list of everyone who has registered, along with their name, contact information, and automobile make and model.*

The next lesson includes a GBS with one additional feature or task, for example a requirement to create two database tables and insert data into one or the other based on an input from the user. The delivery continues in this manner until all specific content areas have been covered to the specified depth. In this way a spiral is created through the content areas as the student progresses

through each successive GBS (see Figure 1). The tier practice courses are designed to progress from one to the next while maintaining a consistent amount of material (albeit more complex) from each content area.

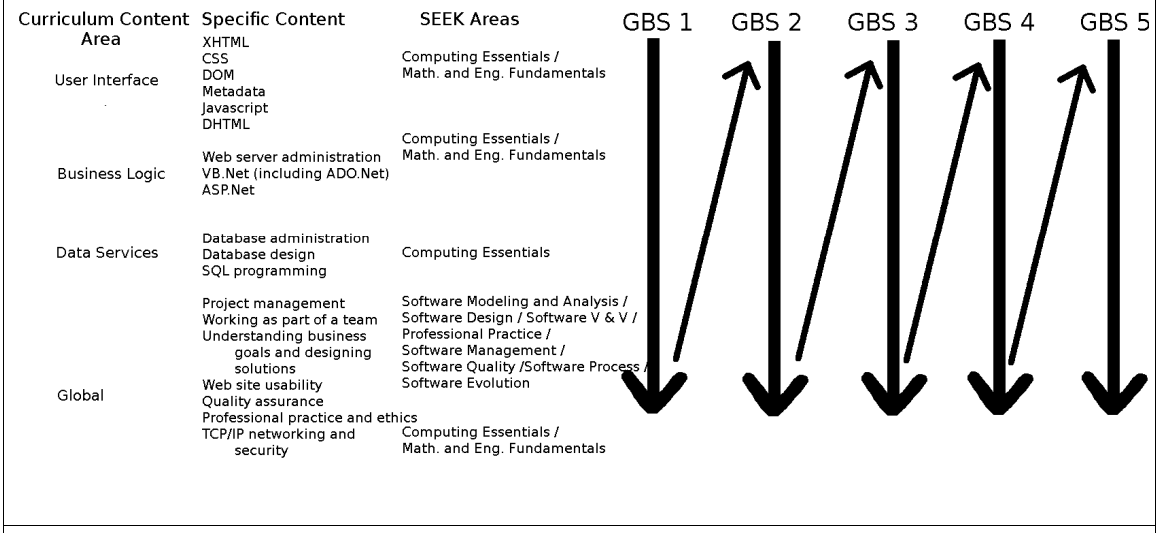

Figure 1. Relative emphasis and distribution of content areas across the curriculum.

Accompanying each tier practice course is a specialty course in which the goal-based scenarios and accompanying instruction tend to focus more heavily on a narrower set of content. The first specialty course, *Programming Concepts*, places a greater emphasis on programming fundamentals using VB.Net and Javascript, understanding the three-tier model, good programming practice, and other basics. The next specialty course, *Computer Networking*, focuses on TCP/IP networking and security, with goal-based scenarios building skills such as Web server configuration and network troubleshooting. The third specialty course, *Database Administration and Programming*, includes goal-based scenarios with an emphasis on the data services tier. *Project Management* follows in the fourth quarter and includes goal-based scenarios requiring group work in which requirements analysis and project planning, understanding business goals, professional practice and ethics, and quality assurance feature prominently. A course entitled *Special Topics in Web Development* follows in the fifth quarter and acts as a capstone course. In this course each student defines his or her own hypothetical business need, proposes a solution, and then develops and implements the solution. Finally, an internship follows in the last quarter in which the student works as part of a Web development team within an external organization. Figure 2 shows the relative emphasis on each content area as an approximate percentage of course content within each course across the curriculum.

Curriculum Content Area	SEEK Knowledge Specific Content Areas	Quarter	1	1	2	2	3	3	4	5
		Course	Introduction to Programming Concepts	Introduction to Server-Side Programming	Computer Networking	Intermediate Server-Side Programming	Database Administration and Programming	Advanced Server-Side Programming	Project Management	Special Topics in Web Development
User Interface	XHTML		2	2	1	1		1	3	5
	CSS		2	2	1	1		1		5
	DOM		2	2	1	1		1		5
	Metadata		2	2	1	1	1	1		5
	Javascript		5	1	1	3		3		5
	DHTML		2	3	1	2		2		5
Business Logic	Web server administration	Computing Essentials / Math. and Eng. Fundamentals			10	9	5	3		7
	VB.Net (including ADO.Net)		35	30	10	30	10	30	5	8
	ASP.Net		30	35	10	30	5	34		8
Data Services	Database administration	Computing Essentials	1	1	5	5	15	2		8
	Database design		5	5	2	5	25	5	5	8
	SQL programming		5	4	3	5	30	10		8
Global	Project management	Software Modeling and Analysis / Software Design / Software V & V / Professional Practice / Software Management / Software Quality / Software Process / Software Evolution	2	2		1	1	1	25	3
	Working as part of a team			2		1		1	20	3
	Understanding business goals and designing solutions		2	2	3	1	2	1	25	5
	Web site usability			2		1		1	2	5
	Quality assurance			2		1	1	1	5	4
	Professional practice/ethics			1		1	1	1	5	1
	TCP/IP networking and security	Computing Essentials / Math. and Eng. Fundamentals	1	1	1	1	1	1	5	2
			4	2	50	1	4	1	5	2

Key: Numbers in cells are approximate percentage; the darker the cell, the greater the relative emphasis is.

0% 5% 10% 15% 20% 25% 30% 35% 50%

Figure 2. Distribution of specific content areas across quarters and courses.

Assessment Issues

As Brown (1998) notes, the constructivist view of assessment is focused on measurement of learning that has value beyond the classroom and meaning to the learner, provides a learning experience in its own right, and is guided by legitimate standards and criteria. As noted by many authors (i.e., MacDonald & Savin-Baden, 2004; Norman, 1998), assessment tools for problem-based learning are dependent on the specific combination of problem solving skills and content knowledge required by the program. What we describe here is our general approach to assessment in order to highlight the elements of our program where assessment is most salient and to indicate that assessment is a consideration we've built into the program from the beginning. However, in this paper we do not consider the many possible methods of assessment discussed in the literature (see MacDonald, 2005, for a review), and we will save a thorough discussion of our assessment experiences and results in light of this literature for a later report.

Student Knowledge Assessment and Progression Readiness

It is important that knowledge assessment techniques maintain the instructional focus on applied skill and real-world scenarios. Knowledge assessments have three important roles to play. First, it is important to confirm that a student has mastered the required knowledge and skill necessary to move to the next GBS. Second, it is important that the instructor identify deficits in a student's knowledge in order to provide appropriate instructional support. Third, and perhaps most importantly, it is critical that rapid feedback on scenario solutions be given such that learners can *index* the information in order to retrieve it later and use it. As described by Schank, Berman, and MacPherson (1999), *indexing* is a critical component of the development of expertise, in which information and experience is organized in memory such that it can be found again and used when needed.

Within a given course (class sizes limited to 20 or fewer students), two basic types of scenarios were assigned on an alternating basis. First was the in-class scenario, completed in an open-lab format in which the instructor would monitor the progress of each student and provide one-on-one coaching as needed. This was purely a mastery-based exercise – the students were required to submit a solution, but could not do so until it was successfully completed. Again, the intention was to provide real-time feedback and coaching on the in-class scenario. Following the in-class scenario was a take-home scenario that the student completed independently. The student was provided detailed feedback on his or her solution prior to being given the next scenario assignment. The take-home scenario was intended to be a more formal test of knowledge and skill acquisition and was thus assigned a traditional numeric grade based on a rubric of the knowledge and skills required to complete it. It was at the same time mastery-based, and the student was not able to progress to the next scenario in the sequence unless mastery had been demonstrated on the previous one. The scenario sequence within each course culminates with a *criterion scenario*, for which a successful solution requires the student to demonstrate all of the learning objectives set for the course.

Because of the mastery-based focus within courses, students were not able to progress to the next course in sequence without successfully completing (and thus mastering the content of) the previous one.

Assessing Program Effectiveness

Among the most important functions of program assessment is to provide feedback for program revision and improvement. From a Kirkpatrick perspective (i.e. Kirkpatrick & Kirkpatrick, 2006), an evaluation of program effectiveness was designed as follows. First, learner reactions are as-

sessed using a brief five-item, Likert-type scenario rating scale given at the end of each scenario (e.g., *The scenario was easy to understand, The scenario helped me learn about Web programming*), along with an open-ended item in which students are asked to make general comments about the scenario. These results allow the instructor to make scenario and other course material revisions where appropriate prior to their next use.

In operational terms, learning is assessed by posing the question *Do graduates of this program have the knowledge and skills identified as program learning objectives by the advisory board?* These learning objectives appear as formal requirements for the student project in the capstone course *Special Topics in Web Development*, as well as the internship, and thus must be met in order to complete the program. For each course, the criterion scenario encapsulates the learning objectives for the course. Therefore the extent to which students complete courses becomes the primary measure of learning. Also, within each course, the instructor evaluates learning of each required content area using a checklist on each of the take-home scenarios. For each student, each content area on which performance is not adequate is recorded. If a pattern emerges for one or more content areas, the previous scenarios in which those areas are covered are reviewed and revised as needed, whether in the current or a previous course.

From an institutional perspective it is difficult to observe changes in behavior resulting from the training provided, in that clearly the goal is for the new behavior to be engaged in outside of the institution in which it was presumably taught. The internship requirement is the one exception to this. A faculty member is designated as the internship faculty supervisor and keeps in frequent, regular, and structured contact with an internship site supervisor. The faculty supervisor maintains a checklist of behaviors the student should demonstrate at the internship site, and this information provides important clues regarding the transfer of knowledge and skills learned in the classroom to on-the-job performance.

Finally, the desired results and ultimate program goal is to produce workforce-ready graduates, and the measure of this is job placement rates. Each graduate is surveyed at three, six, and twelve months following graduation for his or her current job status. Job status is categorized in one of the following ways: employed in field (a majority of job responsibilities fall within the scope of program learning objectives), employed in related field (a minority of job responsibilities fall within the scope of program learning objectives), employed out of field (program learning objectives do not overlap with job requirements), or unemployed. Assuming that the reaction, learning, and behavioral measures are tracking along, a poor job placement rate indicates a re-thinking of learning objectives is necessary.

Closing Remarks

Our primary goal in this paper was to describe our innovations in the development of the current program. It was important for us to situate the curriculum itself within a traditional computer science framework, which we accomplished by carefully considering where the current program overlaps the various areas of subject matter focus among standard curriculum models to identify the best fit. We then sought to identify the best practices and instructional goals for the teaching of software engineering and to identify and implement a delivery methodology that would encompass these goals while meeting the needs of our post-secondary, vocational audience of learners. We believe our program meets the strong school to work bond of vocational education through the application of a spiral curriculum, combined with the constructivist approach of problem-based learning, to place every assignment in every course in a real-world context.

While the curriculum described in this approach is certainly too narrow for a traditional four-year degree program in software engineering, the instructional methods and delivery used may nonetheless be useful in such programs. One can clearly imagine a much broader curriculum than that

presented here being delivered using similar techniques. Although it is beyond the scope of this paper to present it in detail, there is a great deal of contemporary research and theory supporting the use of this approach and providing guidance for its implementation. For example, much has been written about the need to situate learning tasks within real-world contexts (e.g. Lave & Wenger, 1991) the situational and environmental determinants of cognition (e.g. Allen, Otto, & Hoffman, 2004), and the utility of goal-based scenarios and task sequencing to decrease cognitive load (e.g. van Merrienboer et al., 2003), so we will leave it to these and other authors to provide a richer theoretical rationale for the current approach.

Finally, while the degree to which the SEEKs could be thoroughly covered is limited, the current program attempts to follow the ACM/IEEE Joint Task Force on Computing Curricula recommendations as closely as possible and successfully does so in most areas. Further, the instructional techniques used seek to follow sound educational practice in teaching programming skill by developing cognitive schemata in knowledge, strategies, and models.

The program presented here is specifically vocational in its approach and goals. Although this statement is admittedly anecdotal, at the top of the list of students concerns most every instructor in a vocational or "career education" program has heard is something along the lines of *how is this knowledge you're asking me to acquire going to further my career goals?* The answer to this question is explicit in every lesson and every assignment in the program presented here. As such, it should provide a potential model for any instructional endeavor where vocational concerns are paramount.

References

ACM/IEEE. (2002). *Computing curricula 2003: Guidelines for associate-degree curricula in computer science*. (Joint Task Force on Computing Curricula). Retrieved May, 2007 from http://www.acm.org/education/curricula-recommendations

ACM/IEEE. (2004). *Computer engineering 2004: Curriculum guidelines for undergraduate degree programs in computer engineering*. (Joint Task Force on Computing Curricula, Final Report). Retrieved May, 2007 from http://www.acm.org/education/curricula-recommendations

ACM/IEEE. (2006). *Computing curricula 2005: The overview report*. (Joint Task Force on Computing Curricula). Retrieved May, 2007 from http://www.acm.org/education/curricula-recommendations

Allen, B. S., Otto, R. G., & Hoffman, B. (2004). Media as lived environments: The ecological psychology of educational technology. In D. H. Jonassen (Ed.), *Handbook of research on educational communications and technology: A project of the Association for Educational Communications and Technology*. New Jersey: Erlbaum.

Ausburn, L. J., & Brown, D. (2006). Learning strategy patterns and instructional preferences of career and technical education students. *Journal of Industrial Teacher Education, 43*(4), 6-39.

Bennett, J., Grasel, C., Parchman, I., & Waddington, D. (2005). Context-based and conventional approaches to teaching chemistry: Comparing teachers; views. *International Journal of Science Education, 27*, 1521-1547.

Brown, B. L. (1998). *Applying constructivism in vocational and career education* (Report No. RR93002001). Washington, DC: Office of Educational Research and Development. (ERIC Document Reproduction Service No. IN 378).

Bruner, J. (1960). *The process of education*. Cambridge, MA: Harvard University Press.

Cartelli, A., Stansfield, M., Connolly, T., Jimoyiannis, A., Magalhaes, H., & Maillet, K. (2008). Towards the development of a new model for best practice and knowledge construction in virtual campuses. *Journal of Information Technology Education, 7*, 121-134. Retrieved from http://jite.org/documents/Vol7/JITEv7p121-134Cartelli397.pdf

Cheong, F. (2007). Using a problem-based learning approach to teach an intelligent systems course. *Proceedings of 2007 Computer Science and IT Education Conference*, 141-153. Retrieved from http://csited.org/2007/7CheoCSITEd.pdf

Collura, M. A., Aliane, B., Daniels, S., & Nocito-Gobel, J. (2004). Development of a multidisciplinary engineering foundation spiral. *Proceedings of the 2004 American Society for Engineering Education Annual Conference and Exposition, Session 2630*.

Connolly, T., & Begg, C. (2006). A constructivist-based approach to teaching database analysis and design. *Journal of Information Systems Education, 17 (1)*, 43-53.

Connolly, T., & Stansfield, M. (2006). Using games-based elearning technologies in overcoming difficulties in teaching information systems. *Journal of Information Technology Education, 5*, 459-476. Retrieved from http://jite.org/documents/Vol5/v5p459-476Connolly170.pdf

Cornford, I. (1997). Ensuring effective learning from modular courses: A cognitive psychology-skill learning perspective. *Journal of Vocational Education and Training, 49*(2), 237-251.

Deshpande, Y., & Hansen, S. (2001). Web engineering: Creating a discipline among disciplines. *IEEE Multimedia, 8*(2), 82-87.

DiBasio, D., Clark, W., Dixon, A., Comparini, A. G., & O'Connor, L. (1999, November). *Evaluation of a spiral curriculum for engineering*. Paper presented at 29th ASEE/IEEE Frontiers in Education Conference, San Juan, Puerto Rico.

Gentry, M., Peters, S., & Mann, R. (2007). Differences between general and talented students' perceptions of their career and technical education experiences compared to their traditional high school experiences. *Journal of Advanced Academics, 18*(3), 372-401.

Gentry, M., Rizza, M., Peters, S., & Hu, S. (2005). Professionalism, sense of community, and reason to learn: Lessons from an exemplary career and technical education center. *Career and Technical Education Research, 30*(1), 47-85.

Gerjets, P., Scheiter, K., & Catrambone, R. (2004). Designing instructional examples to reduce intrinsic cognitive load: Molar versus modular presentation of solution procedures. *Instructional Science, 32*, 33-58.

Harden, R. M. & Stamper, N. (1999). What is a spiral curriculum? *Medical Teacher, 21* (2), 141-143.

Hilbert, T. S., Renkle, A., Schworm, S. Kessler, S., & Ress, K. (2008). Learning to teach with worked-out examples: a computer-based learning environment for teachers. *Journal of Computer Assisted Learning, 24*, 316-332.

Leitch, S., & Warren, M. (2007). Using problem based learning to teach future Australian IS professionals. *Proceedings of the 2007 Computer Science and IT Education Conference*, 417-424. Retrieved from http://csited.org/2007/8LeitCSITEd.pdf

Linder, S., Abbott, D., & Fromberger, M. (2006). An instructional scaffolding approach to teaching software design. *Journal of Computing Sciences in Colleges, 21*(6), 238-250.

Lave, J., & Wenger, E. (1991). *Situated learning: Legitimate peripheral participation*. Cambridge: Cambridge University Press.

Kirkpatrick, D. L., & Kirkpatrick, J. D. (2006). *Evaluating training programs: The four levels*. San Francisco: Berrett-Koehler.

MacDonald, R. (2005). Assessment strategies for enquiry and problem-based learning. In Barrett, Labhrainn, & Fallon (Eds.), *Handbook of enquiry and problem-based learning: Irish case studies and international perspectives.* Centre for Excellence in Learning and Teaching: Dublin. Retrieved January 1, 2009 from http://www.aishe.org/readings/2005-2/contents.html

MacDonald, R.,& Savin-Baden, R. (2004). A briefing on assessment in problem-based learning. *LTSN Generic Centre Assessment Series.* Retrieved January 1, 2009 from

http://www.heacademy.ac.uk/resources/detail/id349_A_Briefing_on_Assessment_in_Problem-based_Learning

Merrill, M. D. (2002). First principles of instruction. *Educational Technology Research and Development, 50*(3), 43-59.

Norman, G. R. (1998). Assessment in problem-based learning. In D. Boud & G. Feletti (Eds.), *The challenge of problem-based learning* (2nd ed.). London: Kogan Page.

Paas, F. & van Gog, T. (2006). Optimising worked example instruction: Different ways to increase germane cognitive load. *Learning and Instruction, 16*, 87-91.

Reigeluth, C. M. (1999). The elaboration theory: Guidance for scope and sequence decisions. In C. M. Reigeluth (Ed.), *Instructional-design theories and models* (Vol. II). New Jersey: Erlbaum.

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and teaching programming: A review and discussion. *Computer Science Education, 13*(2), 137-172.

Schank, R. C., Berman, T. R., & MacPherson, K. A. (1999). *Learning by doing*. In C. M. Reigeluth (Ed.), *Instructional-design theories and models* (Vol. II). New Jersey: Erlbaum.

Stitt-Gohdes, W. (2001). Business education students' preferred learning styles. *Journal of Career and Technical Education, 18*(1), 32-45.

Trumper, R. (1996). Teaching about energy through a spiral curriculum: Guiding principles. *Journal of Curriculum and Supervision, 12*, 66-75.

Tuovinen, J. E., & Sweller, J. (1999). A comparison of cognitive load associated with discovery learning and worked examples. *Journal of Educational Psychology, 91*(2), 334-341.

Venebles, A., & Tan, G. (2007). A 'hands on' strategy for teaching genetic algorithms to undergraduates. *Journal of Information Technology Education, 6*, 249-269. Retrieved from http://jite.org/documents/Vol6/JITEv6p249-261Venables263.pdf

van Merrienboer, J. J. G., Kirshner, P. A., & Kester, L. (2003). Taking the load off a learner's mind: Instructional design for complex learning. *Educational Psychologist, 38*(1)*, 5-13*.

Whitehead, E. J. (2002). A proposed curriculum for a masters in Web engineering. *Journal of Web Engineering, 1*(1), 18-22.

Yaman, M., Nerdel, C., & Bayrhuber, H. (2008). The effects of instructional support and learner interests when learning using computer simulations. *Computers & Education, 51*, 1784-1794.

Biography

John M. Bunch is President and Principal of The SAJES Group, Inc. He applies instructional science to solve business problems faced by corporate and other organizational clients, in addition to developing and delivering courseware in database and programming technologies. His research interests include the use of problem-based instruction in post-secondary career and technical education, and enhancing learner motivation through instructional design. He spent several years as a hands-on information systems professional and corporate trainer before receiving his Ph.D. in Instructional Technology from The University of South Florida.