Developing Computational Thinking Skills across the Undergraduate Curriculum

Christopher Kuster, John Symms, Christopher May, Chenglie Hu
Departments of Math, Computer Science, and Psychology
Carroll University
100 N East Avenue, Waukesha WI
ckuster@carrollu.edu, cmay@carrollu.edu, jsymms@carrollu.edu,
chu@carrollu.edu

Abstract

In this article, we describe our implementation of the NSF CPATH-supported project that aims to restructure the Bachelor of Science (B.S.) degree requirements and institute a minor and a major in Computational Science. We will describe the design and the initial pilot of the courses that focus on computational thinking. These courses would replace the ones that constitute the current B.S. requirements. We will also describe the design ideas as well as the curricula of the minor and major in Computational Science. An endeavor of this nature with campus-wide impact inevitably brings a number of challenges. We will characterize what the challenges are and how we intend to meet them.

1. Introduction

Wing's influential article [5] suggested that computational thinking is a fundamental skill everyone should have to gain understanding, live, and flourish in today's world. Although the notion of computational thinking remains largely speculative, it is definitive that computing education at all levels of K-16 must be strengthened and improved. The Program of CISE Pathways to Revitalized Undergraduate Computing Education (CPATH) of the National Science Foundation has supported endeavors around the nation that are aimed to improve teaching of computing and promote computational thinking across disciplines. The CPATH-supported project at Carroll University, named Developing Computational Thinking Skills across the Undergraduate Curriculum, was designed to have two primary objectives. First, we will modernize the Bachelor of Science requirements to better prepare our students computationally. Second, we will create a major and a minor in computational science, both of which would companion well with other data-driven disciplines. The project team consists of members from mathematics, computer science, and psychology programs.

The most challenging task has been restructuring and modernizing the B.S. requirements. Currently, the B.S. requirements consist of a course in mathematics (at least at the level of elementary statistics or calculus for non-majors) and a computer science course (at least at the level of a course about Microsoft Office applications). While this combination or a variation of it might be popular in the late 1990s in colleges, it is no longer adequate, nor is in fact appropriate, in addressing increased computational needs in various disciplines. Computation has long become another leg of scientific explorations in addition to experimentation and mathematical modeling. More computationally oriented fields – canonically named Computational X – have emerged in recent years even in such disciplines as art and journalism. However, many professors have not yet included computation as a standard content in their course lessons, nor have they, even in science disciplines, had the knowledge and skill-set to teach computation effectively. Thus, we would provide, by revamping the B.S. requirements, not only students with basic yet fundamental skills of computation, but faculty as well with opportunities to explore the possibility of integrating computation in courses and expand the spectra of their teaching abilities.

Currently, the mathematics requirement of the B.S. requirements is algebra-based statistics (MAT112) or calculus, while the computer science requirement is to take one of the Computer Science courses of any nature. About 80% of B.S. students take MAT112 and CSC107 to satisfy the requirements, where CSC107 is a basic Excel skills course. For our purposes, Computational Thinking and Computational Science are marriages of data analysis, algorithmic design and implementation, and mathematical modeling[1]. The new B.S. courses in Computational Thinking address all three areas, while continuing to meet science faculty expectations in desired skills

[1] Of course, there is not yet an agreed upon definition for "computational science" yet. For example, see http://www.csci.psu.edu/background.html.

(statistical analyses and Excel). Our objective is to equip students with essential data analysis tools, problem-solving ability utilizing the tools, and a broader foundation in computational thinking. The first course focuses on data analysis using statistical means, spreadsheets, and a scientific computing software tool such as MATLAB. We believe that students need to be exposed more with meaningful applications of data analysis with the tools than with the features of the tools themselves. In other words, students would benefit more learning the use of mental tools that endure. We would enhance the current course contents by including a variety of data-analysis applications. The second course in the sequence would tap into one's constructive imagination in solving problems in ways of computational thinking. Different disciplines may uniquely define what computation would mean in the field. Thus, the intent is to create multiple sections of the same course each focusing on an application area of computing. Students would benefit more if they could take a course to learn computation that is domain-specific. The challenge is apparent, however, as most faculty members on campus are not familiar with computation even within their own disciplines. Thus, we would need to create "generic" sections that address computational problem-solving skills that are useful regardless of an applied context. Meanwhile, we hope that our effort, in a long run, would be an impetus for our faculty to establish themselves not only in teaching computation of the field, but in scholarly pursuit as well.

We have piloted the first course in the sequence and had initial evaluation of the outcome. Two versions of the second course with two different themes are being piloted, as of this writing, in the spring of 2011. We will summarize and assess below what we have done in the pilot of the first course. We will also elaborate what we are doing in each of the sections of the second course.

The additional premise of our project is to create a minor first, then, a major in computational science. Admittedly, the current environment is not ideal for fostering such programs. Colleges have endured years of declining in enrollment of computing programs. The College Board has discontinued AP Computer Science AB Exam due to declined interest among high school students. However, we felt that our effort is of strategic importance. Computation is unavoidable not only in the method of study, but also in what is studied [1]. Likewise, computing education is necessary not only to make students more employable, but as part of a well-rounded college education. Carroll University is undergoing a reconstruction of its General Education Program around students obtaining a global perspective and a depth of the knowledge in an area other than their majors. We hope that in the longer term, computing and mathematics will be one of the distribution areas in which students can develop some competency. Meanwhile though, we will need to make these programs we are developing attractive to not only the current students but prospective ones as well. In the following, we will discuss the designs of the proposed courses and the curricula of the proposed programs. Going forward, we are fully aware of the challenges – seeking campus-wide support, recruiting and training instructors, and making the outcomes of our effort endure and prosper. We will conclude the article by describing in detail these challenges that are ahead of us.

2. Restructuring Bachelor of Science Requirements

Computational Thinking I, or CMP112, was piloted in fall 2010, using two separate sections with a combined enrollment of nearly 40 students. Most students in these two sections are natural and health sciences majors. In the spring of 2011, as of this writing, two new sections of Computational Thinking II, or CMP113 and CMP114, are being piloted. Two courses focusing on computational neural science have been previously piloted. One of these courses will become another section of Computational Thinking II. In this section, we will describe our individual efforts in implementing these courses, the course outcomes, and future plans.

2.1 Computational Thinking I – CMP112

The first course in the Computational Thinking (CT I) sequence introduced students to the following topics:

- Function models
- Iterative methods (solving 1-variable algebraic equations)
- Linear and non-linear least-squares regression (single variable)
- Descriptive statistics (measures of center and spread, visualization)
- Probability (discrete and continuous distributions)
- Probabilistic simulation (coins, dice, cards, continuous distributions that are uniform and normal)
- Hypothesis testing (Z-test for ratios, t-tests for means)
- Finite difference models

Each section was limited to 20 students and took place in a computer classroom. Students were recruited for the course through advising, with emphasis on first-year students who traditionally have been unable to take MAT112 (Elements of Statistics) due to limited number of sections and their low priority in needing the course. The attrition rate for both classes was around 10%, similar to the rate for other introductory mathematics courses. Both sections met in a computer classroom, with one section meeting four times per week in 50 minutes periods and the other twice per week in 110 minute blocks.

One of the most successful parts of the course involved the emphasis on modeling and simulation. In the traditional introductory statistics course, students are taught the central limit theorem which explains exactly how taking more data improves knowledge of the population mean. Using simulation to find the standard deviation of the sample mean for different sample sizes, the CT I students were able to fit the model $f(n) = A/\sqrt{n}$ to the standard deviations, showing that the central limit theorem is plausible. Not only did they show the dependence on sample size but they were also able to show that the parameter A in the model was equal to the standard deviation of the simulated population.

The algorithmic thinking objective was measured at twice during the term, the first on the midterm and the other on the final exam. In both cases, students were given a description of an algorithm that they had not been exposed to before and were asked to implement it in Excel. The algorithm tested in the midterm was bisection method to find roots of an equation. The algorithm used on the final exam was the Euler's difference method. The results of the midterm were mixed, with two students correctly implementing the algorithm in Excel, and 50% of the remaining students performing the algorithm using Excel as a calculator but not implementing the logic to automate the process. On the final exam, 65% of the students fully implemented the algorithm showing an improvement in algorithmic thinking skills. Overall however, Student response to the course illustrated strongly a main challenge: delivering the desired content at a level appropriate to students who have not had pre-calculus or calculus. We expect it will take several iterations before this course is at a level that is challenging, yet commensurate with students' preparations.

As a technical reflection, a few changes to the curriculum seem necessary in future pilots. 1) It became evident that the course needs to start with a description of how a computer works. Despite having grown up with computers, most students do not know that there is a difference between the computer's memory and its hard drive, or the limits of what types of computations a computer can do. This information is important, not only for understanding the logic behind computational techniques, but also as part of a modern consumer education. 2) Since the math requirements for the course are relatively low, more time needs to be spent covering arithmetic order-of-operations. Given the emphasis on calculator use in K-12, it was assumed that students would already know this. However, even at the end of the term, there were still students who confused "(1+4)/(7+3)" with "1+4/7+3." 3) Theoretical discussions of a process need to be held away from the computer so that students see them as separate from an implementation of the process. It appeared that the presence of the computer short-circuited the thinking process in many students. For example, when analyzing data, many students would simply copy spreadsheet formulas from work done in class for their use in homework or another class exercise without paying attention to factors such as data location, desired functional form, or even number of data points. 4) The course needs a book. Much of the material was taught from handouts or web-based resources. Since sections of this course will be taught by adjunct faculty in the future, the topics need to be organized in a more formal way. This process is currently underway.

2.2 Computational Thinking II

There are two versions of Computational Thinking II (CT II) – CMP113 and CMP114, and CMP112 is the prerequisite to both. Both courses are being piloted as of this write (spring 2011). The course CMP113 places a greater emphasis on foundational principles of computer science and computing, while CMP114 focuses on developing more advanced data analysis techniques (variations on ANOVA, for example). By allowing the second course to have different emphases, we increase the chance for broad faculty support and buy-in, critical for the project's success.

2.2.1 CT II – Data Analysis Focus (CMP 114)

There are more statistical methods covered in the course with the following topics:

- Nonparametric statistical tests
- Analysis of variance (single/multi-factor, single/multivariate)
- Linear regression and correlation
- Multivariate regression and correlation
- Basic data mining skills
- Basic idea of how computers work

Out of the initial 40 students in CT I, only 7 students elected to take CT II the following term. In part, this was due to limited time in students' schedules and their ability to fulfill the requirements with the existing 2-credit Excel-skills course. All of the students were majoring in a health or natural science. The class met four times per week, 50 minutes per session, usually in a non-computer classroom. The textbook used for the statistics content was [6]. At least once per week, class was held in a computer lab to address algorithm implementation issues. As part of the course, students were required to present reviews of journal articles in their fields. These reviews focused on the hypotheses being tested, the statistical analysis techniques used, and the resulting decisions based on that analysis. At first, students struggled with this assignment, but by the middle of the term, most students were showing interest in reading about findings in their own field of interest and were seeing how the statistical methods covered in class were used in practice.

As of this writing, the course is still in progress. At this point, the only potential change identified for future pilots is a reduction in the planned detail-level to match student ability. Similar to what we have done in the CT I course, we believe that what we have been trying to cover in the course benefits students in improving their critical – as well as computational – thinking ability. The reality, however, is that student inability to handle the rigor we are trying to institute in the course might label, unfortunately, our intent "overambitious". Pedagogy may matter. In future pilots, we will try different pedagogical approaches to facilitate student learning while maintaining the level of rigor we would like to see for the course. Because of the interconnections on computational thinking among different sections of CT II, there is also a possibility in the future that students can go to relevant lectures of a different class to supplement or expand what they would learn in their respective sections.

2.2.2 CT II – Computing Principles Focus (CMP 113)

The other section of CT II being piloted is titled "Introduction to Computing". It intends to achieve two goals: 1) introducing computer science and computing to everyone; and 2) serving as a placement course in the future for the new AP course: CS Principles. In some sense, this course resembles what is called CS0 – a breath-first introduction to computer science – at many institutions. Yet, the difference is that the course we have designed not only introduces students to some essential fields of computer science and computing, but it also focuses on problem solving through programming so that students are exposed to some essential elements of computational thinking. To expose students to the CS principles, we would address in number of ways what information is, what computation is, and how computing has advanced for its own benefit, and how it has influenced the advancement of many other disciplines. Most of students taking this course were born in the Internet age, and they grew up with playing computer games, using hand-held electronic communication devices, and indulging in various kinds of social-networking means. Through this course, we communicate to students not only about the ideas behind computing technologies, but the essence of computation as well.

More specifically, this course introduces students to the discipline of computing by maintaining a balance between computing breath and computational thinking depth. First, it serves to expose students to the field of computing via topics such as the Internet and Web, computer networks, computer architecture, data bases, algorithms, and the history of computing. Second, it provides some depth in two areas that play important roles in fostering computational thinking – programming and design of databases. To effectively address our objectives, we selected a textbook by David Reed [3]: A Balanced Introduction to Computing Science. Students are exposed to major fields of computer science and the depth of programming in alternate classes. Such alternate focus may have some benefit in retaining student interest in the material while keeping them engaged in computing activities at the same time. Another draw for us to use this textbook was the choice of JavaScript as the programming language for learning programming. Despite the obvious drawback of being type-less, JavaScript language produces codes that require nothing more than Web browser to execute. Students can almost instantly be learning event-driven programming without a steep learning curve. For the most part, students would be learning procedurally-oriented programming, thus the poor support of object-orientation in JavaScript has little effect on teaching. JavaScript being type-less may in fact have helped learning as students can focus on problem solving without dealing with type-conformance rules that can be distracting for novices. Microsoft Expression Web, a piece of web-authoring software, provides an excellent editor that supports fully intelliSense for xhtml tagging and

JavaScript coding. Students also learn commonly-used html tags, page layouts, and cascading style sheets along the way.

However, we have designed our course beyond the content of the textbook in multiple areas. On the programming side, students get a feel of programming early on with Scratch – a graphic oriented programming environment [4]. The result was surprisingly encouraging. The first assignment was about the basics of animation, message passing between objects (or sprites), and looping. The second assignment was to create a paper-rock-scissors game that includes a relatively complex (for beginners) if-else structure, as well as variable creation to count the number of wins among other things. The content of the third assignment is at students' choosing. One female student, who is not a CS major, created the grand scene of this year's Super Bowl game, simulating two touchdowns by the Packers with one resulted from an interception (see figure 2.1).

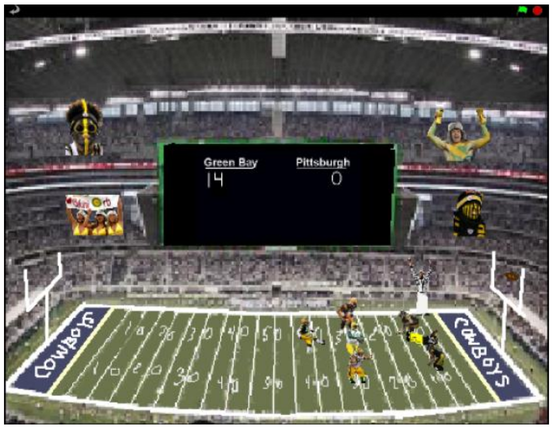

Figure 1: a Scratch program that simulates Super Bowl XLV

Database design is another area we would provide students with depth. We felt that the basic ideas behind database design would help students further develop their logical thinking skills and abstraction ability as data modeling is useful in many applied contexts. Students would use Microsoft Access database management software to create tables and make queries and reports.

Web 2.0 has offered some wonderful opportunities to communicate computing to students. We have used many short video clips from YouTube and other online sources in virtually every field of computing. For example, a 10-minute video would explain how exactly the 'ABC' computer, built at Iowa State College in early 40s, works. Similarly, in less than 5 minutes, student would have some initial, yet important, understanding how Internet, cloud computing, or a logic gate works. Some video clips are also lovely made, effectively capturing viewers' attention. A brief description of course content is the following:

- Scratch programming – 3 weeks
- Week 4 – Introduction of HTML, computer networks, Internet, and Web

- Week 5 – JavaScript basics and dynamic Webpages, history of computers and computing
- Week 6 – Even-driven programming, algorithms and programming languages
- Week 7 – Abstraction and user-defined functions, computing as a discipline
- Week 8 – Websites deployment and conditionals, midterm exam
- Week 9 – Loops and arrays, Data Representation
- Week 10 – Strings, computer architecture & integrated circuits
- Weeks 11 – 13 Database design basics
- Week 14 – Final exam, and Web project due

Our initial observation, as of this writing, is that students were generally enthusiastic about the content, as a number of students typically did more than what the assignments required.

2.2.3 CT II – Computational Neural Science Focus

Dr. Christopher May has taught two courses in the psychology program to pilot material for both a second CT course and an upper level course that can be used in the Computational Science curriculum called "Computational techniques for the Life and Behavioral Sciences". The former would add another dimension in teaching computational thinking that complements ones summarized above.

The first pilot course (another section for CT II) used the overarching framework of complexity theory to introduce students with a limited computational background to a number of computational models. Using the popular and freely available NetLogo simulation package, students explored agent-based models, cellular automata, genetic algorithms, and network science. Each week in a laboratory session, students were assigned two to four models to analyze. Students were tasked with understanding the dynamics of the models by manipulating model parameters and documenting their effects. Using agent-based models, students examined a diverse set of phenomena, including the foraging behavior of ants, flocking of birds, synchronous flashing in fireflies, woodchip gathering in termites, social rebellion, increasing returns in economics, and the distribution of wealth in a society. Students also worked on cellular automata, looking at, for example, how the striping patterns on animal fur can be understood as a 2 dimensional cellular automaton. Using genetic algorithms, students explored the evolution of ethnocentrism and altruism, and co-evolutionary models of arms races in predatory-prey systems. Lastly, students examined the properties and genesis of networks, such as small-world networks and scale-free networks, which characterize many systems from the World Wide Web to the brain. The overarching goal of this course was for students to gain familiarity with systems thinking, thereby coming to appreciate that the behavior of complex systems unfolds over time, is a function of multiple feed-forward and feedback loops, and often exhibits emergence or self-organization.

We have taken two important lessons from these two pilot courses. One, models must be repeatedly tied back to concrete examples in the "real world". This is especially the case for students with a limited computational background. Otherwise, models come to be seen as irrelevant, tedious abstractions. Two, requiring students to go beyond manipulating parameters, and into writing code presents a challenge for those that have no experience doing so. Students sometimes find it overwhelmingly frustrating to be set back for long periods of time because of a missing semi-colon, for example. Students without a programming background may also be less facile with algorithmic thinking. Instructors should be upfront about these challenges, and articulate how the payoff of developing a high quality model far exceeds the struggle that may have gone into creating it. In this regard, computational thinking courses may be particularly well-situated to develop students' intellectual maturity.

3. Design of Computational Science Major and Minor

As of this writing, we have submitted proposals to add a new major as well as a minor in Computational Science and to change our Bachelor of Science requirements. We will address questions and concerns posed by the school's Academic Steering Committee, and hope that the proposals will receive favorable faculty-wide votes of approval.

Our chief focus was to develop a minor that would support well any data driven discipline. In order to make the minor as broadly attractive as possible for students, we needed to balance the need for developing skills in mathematics and computer science with the need to attract students in areas such as psychology and biology. In other words, the more mathematics and computer science courses required within the minor, the fewer students that would actually choose the minor. Thus, the minor requires a single semester of calculus, a single semester of intermediate programming, and the two Computational Thinking courses (that B.S. students would be required to take). Additionally, we have developed two application-oriented Computational Science courses, and taking one of the two is the final requirement of the minor. The proposed minor is summarized in the following table.

Computational Science		
Computational Thinking I	CMP112	*(4 Credits)*
Computational Thinking II or	CMP113	*(4 Credits)*
Computational Thinking II – Data Analysis Emphasis	CMP114	
Computational Techniques for Life and Behavioral Sciences or	CMP330	*(4 Credits)*
Computational Techniques for Physical Sciences	CMP340	
Computer Science		
Introduction to Java or	CSC111	*(4 Credits)*
Advanced Programming in C#	CSC112	
Mathematics		
Calculus and Applications or	MAT140	*(4 Credits)*

Calculus I	MAT160	

Table 1: Minor in Computational Science (20 Credits)

The proposed major in computational science has moderately demanding math and computer science components with more emphasis on modeling and simulation. We understand that differential equations underlie many computational models, and thus a course on differential equations seems essential. However, at this point, we intend to briefly introduce differential equations in CMP330 and 340 as part of course content as opposed to making a differential-equations course a separate requirement. The following table 3.2 summarizes the course requirements.

Core (40 Credits)		
Computational Science		
Computational Thinking I	CMP112	*(4 Credits)*
Computational Thinking II or	CMP113	*(4 Credits)*
Computational Thinking II – Data Analysis Emphasis	CMP114	
Computational Techniques for Life and Behavioral Sciences	CMP330	*(4 Credits)*
Computational Techniques for Physical Sciences	CMP340	*(4 Credits)*
Computer Science		
Introduction to Java or	CSC111	*(4 Credits)*
Advanced Programming in C#	CSC112	
Data Structures using Java	CSC226	*(4 Credits)*
Database Design	CSC351	*(4 Credits)*
Mathematics		
Calculus I	MAT160	*(4 Credits)*
Calculus II	MAT161	*(4 Credits)*
Linear Algebra	MAT208	*(4 Credits)*
Other Requirements		*(16+ Credits)*
Students must minor or major in a discipline in which a bachelor of science degree can be earned.		

Table 2: Major in Computational Science (Minimum 56 Credits)

The two new Computational Science courses, CMP330 and CMP340, will contain a wide variety of computational models from disciplines including biology, psychology, physics, astronomy, chemistry, biochemistry, bioinformatics, finance, sociology, ecology, and health science. Students choose the course that contains the applications of their interest. We expect small numbers in the major, but are hopeful for solid numbers in the minor.

The second course Dr. May piloted was an upper level course on Computational Neuroscience, and can be potentially tweaked for CMP330. As Dr. May notes in his course description, "Brains are composed of ~10 billion interconnected neurons, operating in parallel. As such, brains cannot be understood by thinking in terms of simple sequences of events. Understanding the brain at a deep level requires studying and manipulating models grounded in principles of brain function." In this course, students stepped through a large number of neural network exercises, detailed in [2]. Students used a free neural network platform called Emergent, which was created by O'Reilly. Through these models, students examined a diverse set of phenomena, including categorization, pattern completion, generalization, constraint satisfaction, and supervised and unsupervised learning, in order to better understand the nature of attention, memory, language, and higher-order cognition in humans. This course employed an integrated lecture-lab format, such that students were continuously working to understand some model. For the majority of models, rather than manipulate parameters, students sought to understand why different input patterns would lead to different outputs, given the configuration of the network. Similar to the course on complexity theory, an overarching goal of this course was for students to practice systems thinking in order to develop a better understanding of complex systems. Whereas the CT II course surveyed many different systems, this course focused on the brain.

4. Challenges ahead

From an administrative standpoint, the biggest challenge is staffing the Computational Thinking courses (CT I and CT II). The course MAT112, which constitutes part of the current B.S. requirements, is a standard non-calculus-based introductory statistics course. There are plenty of people in the region with the credentials, experience and desire to teach the course. The Computational Thinking courses, however, are not standard. In particular, we know of no textbooks that fit the design of the first course – a , CMP112. This makes staffing the new curriculum challenging, and will require finding some part-time instructors willing to undergo training. However, in addition to facing staffing challenges, we may also face challenges of getting realistic support from faculty who might be potentially developing computationally-oriented contents or courses to meet the needs of improving the proposed programs. Any faculty endeavors of this kind would require not only the willingness, but also the dedication and even re-orientation of their scholarship pursuits. Thus, support from senior school administers is paramount in order for our effort to be sustainable.

As commented prior, the continued revising of CMP112/113/114 so that content is at an appropriate level is a critical challenge. If the courses are not done well, the faculty will likely (and understandably) remove or change the B.S. requirements. The Carroll faculty deserves praise for supporting mathematics and computer science components in the education of all B.S. students, and we don't want to undermine that support.

The final challenge is controlling ourselves with respect to the amount of mathematics and computer science in the minor. Each revision of the minor has resulted in removal of more mathematics and computer science courses (initially we had Calculus I through Numerical Analysis in the minor, which would not have been popular in the life sciences). Other institutions[2] have tried to build similar minors, but they included too many mathematics and computer science courses, thus creating minors that did not attract an outside audience. Removing the mathematics and computer science courses from our minor to its current state was emotionally painful, however, we hope to now have a minor that will attract a heterogeneous audience. Fostering a computing education in our school (and beyond indeed) requires patience. The computational programs we have designed might not be the ones we wished we would, but are reasonably practical to get them off the ground. These programs will inevitably improve as the academic climate improves.

References

[1] Denning, P. Beyond Computational Thinking, *Commun. ACM*, Vol. 5, No. 6, June 2009, 28-30.

[2] O'Reilly, R. and Munakata, Y. Computational Explorations in Cognitive Neuroscience, The MIT Press, Cambridge, Massachusetts, 2000.

[3] Reed, D. *A Balanced Introduction to Computer Science*, 2nd Ed., Prentice Hall, Upper Saddle, NJ, 2008.

[4] Scratch – Imagine, Program, Share, at http://scratch.mit.edu/.

[5] Wing, J. Computational Thinking, *Commun. ACM,* 49, 3 (March 2006), 33-35.

[6] Zar, J. Biostatistical Analysis, 5th edition, Prentice Hall, Upper Saddle River, NJ, 2010.

Acknowledgment

The authors would like to acknowledge the support of NSF CPATH program No. 0939032.

[2] http://www.uwec.edu/registrar/Catalogues/current/multdisc.htm; http://www.uwlax.edu/sah/html/undergradprograms.htm