Computing Foundations for the Scientist

Catherine Bareiss
Department of Computer Science
Olivet Nazarene University
Bourbonnais, IL 60914
cbareiss@olivet.edu

Larry Vail
Department of Computer Science
Olivet Nazarene University
Bourbonnais, IL 60914
lvail@olivet.edu

ABSTRACT
There is a need for a new style of supporting a computer course. Although it is widely recognized that computer technology provides essential tools for all current scientific work, few university curricula adequately ground science majors in the fundamentals that underlie this technology. Introducing science students to computational thinking in the areas of algorithms and data structures, data representation and accuracy, abstraction, performance issues, and database concepts can enable future scientists to become intelligent, creative and effective users of this technology. The intent of this course is not to turn scientists into computer scientists, but rather to enhance their ability to exploit computing tools to greatest scientific advantage.

1. INTRODUCTION

Aided by powerful software packages, scientists today build complex visual system models, manage scientific databases, perform simulations, consult expert systems, and render results. If science students can understand this software as more than a black box, they can be positioned to better understand its value and results, and make more intelligent decisions about how to analyze and improve their results, and even when to rely on the software and when not.

The purpose of this course is to make future scientists more intelligent users of computing technology in their practice of science. At ONU an introductory course on computing foundations and their specific application to areas of science has been developed. The material in this course been developed in at least 14 modules that can be combined into one course, used as part of existing science courses, and used by students for independent learning.

2. BACKGROUND

The need for understanding of computational thinking in today's society by all disciplines is well documented [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]. According to Peter Denning [6], "many whose lives are touched by computing want to know how computers work and how dangerous or risky they are, . . . and most everyone asks for an uncomplicated framework for understanding this complex field."

It is essential to understand what computational thinking is. It is understanding the fundamental concepts of computer science and applying them to most every area of life. According to Denning [6], this includes computation, communication, coordination, automation, recollections (windows of computing mechanics) and simplicity, performance, reliability, evolvability, and security as design principles. Not only is computational thinking conceptualizing and not programming[6], Lu [9] explains that programming should not be a student's introduction to computational thinking, just as proof construction is not a person's introduction to mathematics (arithmetic is). "The emphasis should be on understanding

(and being able to manually perform) computational processes, not on their manifestations in particular languages." Courses based on this should address properties such as "convergence, efficiency and limits of computation."
Computational thinking is foundational to the scientist [5, 7, 8, 10, 11, 12]. Nobel Physics Laureate Ken Wilson stated that "computation had become a third leg of science, joining the traditions of theory and experiment [5]." Hambrusch [8] states "scientific research is now unthinkable without computing. The ubiquity of computerized instrumentation and detailed simulations generates scientific data in volumes that can no longer be understood without computing."

Many have understood the charge to promote more computational thinking in disciplines outside computer science to mean the introduction of programming in these non-computing curricula. At a SIGCSE 2009 panel on the present and future of computational thinking, three leaders in this area (Astrachan, Hambrusch, and Settle) reported the inclusion of programming in non-computing courses [2][8]. A multi-disciplinary approach at the University of Nebraska [11] also works with biology (and some non-science disciplines), but still incorporates programming.

One current approach [9] covers computational thinking with biology students without programming, but is aimed specifically to biology majors at the junior/senior level, which precludes the possibility of adding a computing minor or cross pollination of ideas across disciplines. At Carnegie Mellon University [4], there is a course on computational thinking that does not include programming, but this course is designed for the general student population and does not focus on the areas specific to the sciences.
This course is unique in its effort to more broadly define computer literacy and fluency for the science student, and to focus on coverage of a variety of non-programming computing concepts that will enrich the science student's appreciation of computer technology as a valuable tool to be used in the creation of science.

3. WHAT IS IN THIS COURSE?

The Computing Foundations for the Scientist course has been developed at Olivet Nazarene University as the product of an NSF CCLI grant as a possible supporting course for most science majors. It has been designed to teach fundamental concepts of computing that are essential to scientists as they do their work. Details about the course can be found at the following URL.
http://cs.olivet.edu/twiki/bin/view/ComputationalScience/WebHome

This course has three major goals:
1. To help science majors understand the benefits and limitations of the technology they use
2. To equip science majors to better optimize the use of software in future courses
3. To help science students see connections between the many different sciences that they might not experience as they focus on their own particular major

To help the students understand the benefits and limits of technology, these are explained and demonstrated through the use of different science examples. For example, when XML is discussed, an XML document of the periodic table is used. To help them optimize the use of software, different software is used to solve different types of science problems, including simulation software, GIS software,

discipline specific websites, and spreadsheets. By using these (and other tools) to solve specific science problems, students see new and better ways to look at different problems. By taking specific, real life topics from many different science courses, students start to make connections between different sciences. Biology students can see that simulating population growth is similar in principle to simulating chemical reactions. Engineers can see that studying fluid dynamics is similar to some of the things geologists study.
This course was developed in a modular format with four modules explaining the background computing concepts and ten modules taking those concepts and showing how they impact different areas of science and math. They are structured in such a way that the ordering is flexible (as long as the prerequisite knowledge is covered first) and easily expandable with new computing modules and new science modules

The areas of computing covered by the different modules include: algorithm understanding, data accuracy and the source of errors, performance issues (including Big-O), reliability, simulation, visualization, abstraction, databases, data structures, and storage issues. These topics (and others) were chosen based upon their impact on the different science topics that were chosen. Other topics might be necessary if different topics where used.

The modules developed are as follows:

CSIS 1	Introduction to Computational Science
CSIS 2	Data types: Representation, Abstraction, and Limitations
CSIS 3	Procedures: Algorithms and Abstraction
CSIS 4	Self -Defining Data: Compression, XML, and Databases
BIOL 1	Bioinfomatics
BIOL 2	Cladograms
CHEM 1	Chemical Kinetics
CHEM 2	Molecular Modeling
ENGN 1	Design and Analysis with Engineering Spreadsheets
GEOL 1	Geographic Information Systems and Spacial Analysis
GEOL 2	Flow Analysis
MATH 1	Solving Equations
MATH 2	Curve Fitting
NSCI 1	Scientific Data Acquisition

Details on each of these modules (and future modules) and the course in general can be found at: URL: http://cf4s.olivet.edu.

The computer science modules were written by computer science professors. The biology, chemistry, engineering, and geology modules were written by biology, chemistry, engineering, and geology professors (respectively). The remaining modules were written by teams from different disciplines.

This course has been taught by one of the computer science professors using active learning techniques. Minimal time was spent lecturing (unless there was a complex issue that everyone needed help understanding). Instead the students (working in pairs) spent much of the class time reading the modules and working through the examples and questions embedded in each module.

Each module starts with an overview of the concepts being studied. It then gives a short lesson on a concept in that module followed by an on-line activity and questions. This repeats until the module is finished. At the end of the course, students present projects that they have developed on their own using some of the skills learned in the course. The individual module assignments, project, and exams combine to form the grade for the course.

4. WHY CONSIDER TEACHING THIS COURSE?

4.1 The Benefits We Experienced

There are four major benefits we experience from this course (besides additional benefits that came from the grant itself). The biggest benefit was a course that is designed to better meet the need of most SEM (science, engineering, and mathematics) majors. Many existing supporting courses fall into one of two categories: learning to program or learning to use specific software (such as MATLAB). While there are good reasons for these courses, most scientists do not write programs any more. They either have complex software that they can use or will work with a computing professional to develop very specialized software. In addition, being able to use software is not good enough for the scientists. They need to be able to understand the results, know if the results can be trusted (why or why not), and realize what technology can and cannot do for them. This course is better designed to meet these goals.

The second benefit that came from the course was experienced by the professors and students alike. Each of us learned a lot of science that was outside our disciplines. In addition, the SEM professors received a much better understanding of the underlying computing concepts and have been able to incorporate this knowledge into some of their advanced courses.

The third benefit was experienced by the computer science students that took this course. In many computer science courses, students learn the concept removed from the applications that use it. While we may discuss these areas in class, they have limited personal exposure to these areas. Computing students taking this course get a strong introduction into different ways computer science in used in the sciences. This has expanded their horizons to see additional areas that they might want to study and/or work in.

The fourth major benefit this course has brought to ONU is increasing the awareness that most students need a better understanding of technology. Being capable to use the computer for personal needs (such as word processing, socializing, research, email, etc.) is not sufficient in today's society. Most disciplines use technology in a very advanced way (and this is only going to increase). It is important for the users of technology to not treat it as a "black box" and accept the results without question. It is also important for them to know what technology is capable of and what is not. This course demonstrates such things to the SEM community on campus and is being used as a model when talking with other disciplines.

4.2 Additional Reasons

There are a number of additional reasons why a department might consider adopting a similar course. The first two reasons deal with costs. All the software used in this course is either free/public domain or already on the campus (Microsoft Excel). There is no need to invest in software. In addition, the modules themselves are free (to both the department and students). A department may take the modules and teach the course without much additional work. The students don't even need to buy a textbook!

One thing that might discourage someone from teaching such a course is limited expertise in the SEM areas. Being strong in the sciences is not necessary to teach this course. The modules are written so that sophomores in different areas can understand the science. So an engineer who has yet to study any biology can understand the biological modules. The biology student can read and understand the geology areas. The math students can following the chemistry covered. What is required is an understanding of the scientific method and the ability to use mathematics. (While there is some calculus in some of the sciences, students aren't even required to be able to do the calculus.)
A third reason to consider adopting such a course is that it is easy to expand. If an institution wants to expand the topics into another discipline (such as physics), this can be done easily. The existing computer science modules can be used (and more written if necessary) and a physics module can replace one of the science modules. If the institution has a researcher very involved in another area of chemistry and wants to include a module in that area, that researcher needs to just follow one of the existing templates to write a new one. As more and more institutions adopt such a course, there will be a wide variety of modules to choose from and to even vary the course from year to year.

The last reason to consider adopting such a course is probably the most important one. As one thinks about the future of computer literacy, we need to adapt from the old models as technology changes and how we use it. Very few people (other than computing professionals) write any major code. But when they use programs, there are some concepts that they need to be aware of. These concepts can be learned via programming (such as performance found in Big-O understanding) but can also be learned other ways. By focusing on the details of computing that are essential, time can be spent on helping the students apply them in areas that they will realistically encounter in their professional life.

5. WHAT NEXT?

There are four areas of additional work for this course. This first area will be done at ONU over the next 18 months. As the course is further refined, it will be taught again and additional formal assessment will be done. This will help us as we add modules (three are already planned), better connect the modules to each other, and find other ways to improve the overall experience.

The next area is to work with other institutions that might want to adopt this course. Such institutions need to be identified and resources. In addition, as other institutions use these modules, additional improvements will be identified.

As this work progresses and expands, additional science disciplines will contribute more modules and additional topics for existing disciplines will be added. More computer science modules will be added as needed. All new modules will be added to the repository.

The last area of future work is once again probably the most important. The philosophy of this course can be expanded beyond the sciences. Just as there are fundamental concepts about computing that are necessary for the scientists to understand so that they can do their work well, there are fundamental concepts that are essential for those in the health care industry, the artist, the communicator, the historian, and the educator (just to name the few). Similar courses can be developed for many different areas. The computer science modules can be used (and adapted to use different examples) as needed and additional ones can be used. The goal will be for all students to be able to learn the computing concepts that are important for them to succeed in their discipline. This need will only become greater as each discipline demands more and more complex technology to support them in their areas!

6. REFERENCES

[1] Adams, Joseph Brian. Computational Science as a Twenty-First Century Discipline in the Liberal Arts, Journal of Computing Sciences of Colleges, Vol. 23, Issue 5 (May 2008), pages 15-23.

[2] Astrachan, O., Hambrusch, S., and Settle, A., The Present and Future of Computational Thinking, SIGCSE Bulletin, Volume 41, Issue 1(March 2009), pages 549-550.

[3] Cohen, Avi and Haberman, Bruria, Computer Science: A Language of Technology, SIGCSE Bulletin, Vol 39, Issue 4, pages 65-69, 2007.

[4] Cortina, Thomas J., An Introduction to Computer Science for Non-majors Using Principles of Computation, Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education, pages 218-222, 2007.

[5] Denning, Peter J., Computing is a Natural Science, Communications of the ACM, Vol 50, Issue 7 (July 2007), pages 13-18.

[6] Denning, Peter J., Great Principles of Computing, Communications of the ACM, Vol 46, Issue 11 (November 2003), pages 15-20.

[7] Easton, Thomas A., Beyond the Algorithmization of the Sciences, Communications of the ACM, Vol 49, Issue 5 (May 2006), pages 31-33.

[8] Hambrusch, S., Hoffman, C., Korb, J, Haugan, M., and Hosking A, A Multidisciplinary Approach Towards Computational Thinking for Science Majors, SIGCSE Bulletin, Volume 41, Issue 1(March 2009), pages 183-187.

[9] Lu, James and Fletcher, George, Thinking about Computational Thinking, SIGCSE Bulletin, Volume 41, Issue 1(March 2009), pages 260-264.

[10] Qin, Hong, Teaching Computational Thinking through Bioinformatics to Biology students, SIGCSE Bulletin, Volume 41, Issue 1(March 2009), pages 188-191.

[11] Soh, Leen-Kiat, Samal, A., and Scott S., Renaissance Computing: An Initiative for Promoting Student Participation in Computing, SIGCSE Bulletin, Volume 41, Issue 1(March 2009), pages 59-64.

[12] Wilson, G., Alvarado, C., Campbell, J., Landau R., and Sedgewick R., CS-1 for the Scientist, Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education, pages 36-37, 2008.

[13] Wing, Jeannette M., Computational Thinking, Communications of the ACM, Vol 49, Issue 3 (March 2006), pages 33-35.

7. ENDNOTE

This material is based upon work supported by the National Science Foundation under Grant Number (0942362). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.