Content + Experiences = Curriculum

Judith L. Gersting
Department of Computer Science
University of Hawaii at Hilo
Hilo, Hawaii 96720 USA
gersting@hawaii.edu

Frank H. Young
Department of Computer Science
Rose-Hulman Institute of Technology
Terre Haute, Indiana 47803 USA
young@cs.rose-hulman.edu

1. Introduction

The academic curriculum in computer science has been proposed, reviewed, and modified in an on-going process for years [1, 2, 3, 4]. The resulting curricula have packaged the subject matter of computer science in many ways. The end result of these studies has been a carefully designed coverage of academic material in a fashion that requires students to master core material and guarantees student exposure to appropriate additional material that insures both breadth and depth of knowledge.

We suggest that describing the "material that must be covered" is an inadequate approach to curriculum design, necessary but not sufficient. Whether one is teaching in a liberal arts environment or in a more pre-professional environment, there are more aspects to an education than the content that is covered in required and elective courses.

In this paper we propose an experiential aspect of the computer science curriculum as a complement to the content aspect. We urge those who are designing new curricula or revising existing ones to consider this aspect and evaluate their curricula with regard to experiences as well as the more usual content evaluation.

2. The Case for Experiences

Academic content defines the structure of a computer science curriculum. But in addition to academic content, there are a number of experiences that an undergraduate computer science student should have somewhere during the course of his or her program. These experiences are important, we believe, because (as with academic content) they will better prepare the student for both initial and long-range success in the workplace environment. However, a list of appropriate required experiences has not received the scrutiny that the list of required content has. Few departments have formalized such a list. Even if we were all to agree on the experiences to include, their placement in the academic curriculum is not well-defined. As a consequence, it is easy for them to fall through the cracks with the result that a student may graduate minus most of them.

The purpose of this paper is to stimulate discussion about desirable experiences for the undergraduate computer science major by proposing a draft list (Section 3). Depending on the requirements of a particular degree program, students will already have some of these experiences, many in a software engineering course. Some experiences may be found in courses outside of the department. A few items on the list may even be somewhat controversial, or difficult to arrange.

The items on this list sound like assignments. However, unlike ordinary academic assignments, the process of undertaking them is more important than their successful completion. Students will change as a result of these experiences, and that change measures the value of the experience more so than the quality of the product or outcome. This means that when items on the list are assigned, there should be time reserved to reflect on and evaluate the experience. Assessment of a student's participation in the experience should include consideration of the student's own reflections and evaluations. This represents, to most of us as educators in the sciences, a rather novel approach to "grading."

As you read the following draft list, consider those experiences you believe to be important, or others you would add to this list. Think of where they are now being met or where they could be included in your own degree program. Some suggestions follow in Section 4.

3. A Draft List of Experiences

Experiences that develop evaluative abilities.

Read code written by another person and evaluate its internal documentation, test whether it satisfies stated specifications, determine whether stated code standards were observed, and prepare a written summary evaluation of the code.

Critique another student's resume.

Critique the documentation of a commercial software application.
Edit/review/referee/analyze/annotate someone else's written work.
Analyze competing hardware or software products for possible purchase and evaluate some possible vendors.
Analyze the social, economic, and legal impact of a proposed project upon a community and write a position paper based upon this analysis.
Study a failed real-world project and attempt to pinpoint the cause(s) of failure: technical, economic, human, etc., making suggestions that might prevent such a failure in the future.
Evaluate the user interface of a commercial product, of a program written for another course, and of a system produced as a student project.
Design an interface to a tool or system.
Analyze and critique the methodology used to solve a specific problem.
Prepare a written performance evaluation of a co-worker and/or an employee.
Perform some audience analyses, e.g., for some documentation, for a proposal.
Perform a market analysis for a new product.
Summarize a technical article.
Review a request for funding, e.g., a research proposal, a request for additional staff.

Experiences that develop writing abilities.
Prepare an assignment description, a laboratory writeup and several examination questions.
Compare and contrast some related technical articles.
Prepare a memo that presents basic information about a topic, for example, what is known, what is current practice, what changes are likely soon.
Write a paper describing where you hope to be in your work environment ten years after graduation and what you think it will probably take to get there.
Write a memo comparing and contrasting some possible solutions to a problem.
Write a technical recommendation supporting a course of action such as the purchase of a product, a decision to continue or drop a project, etc.
Write a paper that tracks the "soft requirements" that will impact the success or failure of some proposed project: the company policy, the "political" lay of the land with respect to the project, who must approve it and their biases, etc.
Prepare a report on accepted professional procedures/conduct when working as a consultant.
Write a letter to the editor that involves the presentation of some technical information.
Translate some verbal instructions into written form.
Write instructions for using a tool/machine/program to do a given task.
Prepare a set of standards to be used in some context.
Create some advertising copy and/or a press release for a product.
Prepare a poster presentation about a project/assignment.
Prepare a technical paper for submission to a publication.
Prepare a progress report for a superior, e.g., a short memo, a formal report, a final report.
Prepare a job description.

Experiences that develop abilities in group and project work.
Work on a team project.
Serve as team leader for a team project.
Coordinate the activities of two or more teams working on different parts of a single project.
Lead/facilitate a group discussion.
Prepare the agenda for a meeting.
Preside at a meeting.
Take and prepare minutes of a meeting.
Present a position paper to a meeting.
Prepare an RFP.
Prepare a project proposal.
Prepare a feasibility study and a budget estimate for a proposed project.
Prepare a schedule or work plan for a project, including all needed resources.
Interview a customer/client to determine what the customer/client wants, prepare a written summary, and have the summary reviewed by the customer/client for correctness.
Divide a task up into parts, assign the parts to different members of a team, monitor the progress of the work, and make sure that the completed parts can be combined to accomplish the original task.

Experiences that develop respect for the work and skills of other people.
Perform routine system maintenance tasks for a PC and a workstation, e.g., install an operating system from scratch, upgrade an operating system, install a large application, upgrade a large application, uninstall an application.
Open up a PC and a workstation to see what the components actually look like.
Clean accumulated dust and grime from the inside of an old computer.
Upgrade the memory of a computer including determining the kind of memory needed, identifying possible vendors, obtaining prices, arranging for purchase, and actually installing the

memory (both with an old machine and a new one).

Replace and/or upgrade a computer's hard drive including evaluating the appropriate size in light of available funds, identifying possible vendors, investigating the appropriateness of used equipment, obtaining prices, arranging for purchase, and actually installing the drive in the machine.

Actually run some networking wire from one place to another, e.g., between buildings (through a conduit), between offices (through walls and ceilings).

Network two computers, including determining, installing, and configuring necessary hardware; determining, obtaining, and installing necessary software; and verifying the correct behavior of the resulting network.

Write a shell script or a batch file.

Experiences that foster personal development and maturity.

Attend a formal dinner (menus/ordering, dining etiquette, tipping...).

Host a formal dinner at a good quality restaurant.

Arrange and coordinate a telephone conference call.

Plan for and take a business trip involving air travel and a hotel stay.

Have a practice job interview.

Have work performance reviewed by a supervisor (or equivalent).

Prepare a resume for submission to a group of evaluators.

Do something completely, 100% correctly - pass a test, write a program, prove a theorem, etc.

Do something that is initially believed impossible to accomplish - solve a problem, write a program, etc.

Attend a local meeting of a professional society.

Attend a national technical conference (better yet, help host one).

Spend at least one week working on a job with uncooperative and antagonistic coworkers.

Spend at least one week working under an incompetent and unreasonable boss.

Prepare a professional development plan, including continuing education activities.

Experiences that develop presentation and interaction abilities.

Prepare and give several types of formal presentations, e.g., using PowerPoint or an equivalent, using a blackboard or whiteboard, using poster sheets, using a hand-out, using no presentation aids at all.

Present an outline/explanation of a problem solution for review.

Present a report on a project/assignment to a group of outside visitors.

Present a formal lecture on a technical topic to a technical class.

Present a formal lecture on a technical topic to a non-technical audience.

Mentor a beginning student.

Read and present the results of a technical paper.

Read a group of technical papers on one topic and give a presentation that summarizes and evaluates their contribution to the topic.

Train a novice user to use the basic user interface of an operating system.

Train a novice user to use software the student knows well.

Train a non-technical person to use a somewhat technical software package.

Attempt to convince a dyed-in-the-wool FORTRAN programmer of the benefits of the object-oriented approach. :)

Document someone else's product (perhaps not CS-related).

Write instructions for doing a standard system maintenance task (backing up the hard disk, changing the hard disk partitioning, reinstalling the operating system, recovering files from an old backup, etc.) and have someone else (a novice!) do the task by following the instructions. Then revise the instructions as needed!

Plan an employment interview.

Interview a candidate for employment.

Conduct a face-to-face review of the work performance of a person you have supervised.

Make an extemporaneous progress report to a supervisor, e.g., in the hallway on the way back from the coffee machine.

Teach something to a coworker/peer.

Learn something from a coworker/peer.

Participate in a meeting run under "Robert's Rules of Order", including making a motion, arguing for a motion, arguing against a motion, and making an amendment to a motion.

Experiences that develop the ability to obtain and use resources.

Search for the answer to a technical question using the World Wide Web.

Search for the answer to a technical question using usual library resources.

Search for the answer to a software question using the documentation that was provided with the software.

Research technical information in an application area somewhat removed from CS.

Prepare an annotated bibliography on a certain topic.

4. Integrating Experiences into the Curriculum.

Once a list of required experiences is developed, attention must be given to actually fitting these experiences into the academic framework. Faculty will be unwilling to sacrifice much time in an already crowded syllabus. Some practical suggestions follow.

Suggestion 1: Don't try to put all of these experiences into the software engineering course. Many of the experiences listed above are appropriate for software engineering classes. Instructors who teach software engineering have little problem incorporating these experiences into their courses. Their only difficulty is that there are too many experiences to incorporate into that one course.

Suggestion 2: Introductory courses need experiences too. Curriculum designers who are considering integrating experiences into an undergraduate curriculum should attempt to distribute the experiences over the whole curriculum. Many of the experiences we have suggested will help introductory students develop attitudes that will facilitate their success in upper division courses as well as their workplace environment after graduation. These experiences are too important to be saved until the upperclass years of an undergraduate program. For some institutions, introductory computer science courses serve both majors and nonmajors, but students in other majors will also benefit from taking part in these sorts of experiences.

Suggestion 3: Build on previous experiences. For example, reading and presenting the results of a technical paper (a presentation experience) can precede summarizing a technical article (an evaluation experience). A student who works on a team project can serve as a team leader in a later project.

Suggestion 4: Use extracurricular venues for some experiences. A student club or student professional society, for example, can provide a setting for group experiences, hardware repair projects, etc.

The following is an account of some of the ways these experiences can be incorporated into introductory computer science courses.

A. Have a final project in CS1 and/or CS2 that is a group project. Include as part of the project both an oral presentation and some form of advertising. The oral presentation is to be a sales presentation that describes the advantages and special features of the product. The advertisement can be posted on the instructor's office door or bulletin board (this will stimulate student creativity!). This will encourage introductory students to incorporate interesting user interfaces and optional features into their products. Thus, beginning students are encouraged to think in terms of the total quality of the product and also must evaluate the effort required to incorporate that quality into the product.

B. Ask students to modify code written by another student. This requires students to read code written by another person as well as determine whether it actually works and find any errors. In our experience students will complain loudly about the quality of the code they must read and will be much more receptive to suggestions about improving the quality of their own code.

C. In those institutions where introductory courses are reasonably small (25-30 students) it is possible to handle the introduction to a project design as a group discussion. It can be useful for such a discussion to be led by a student.

D. In some beginning courses there are closed laboratories. In such a laboratory it is very useful to have periodic checks of student work. When students reach a certain point they are asked to have their work reviewed to make sure that they have done things correctly. One can adopt the practice of having these reviews done by students whose work has previously been reviewed. This is an excellent experience for the reviewing students.

E. In introductory courses one can create a local newsgroup so that students can engage in discussion and get questions answered using a newsreader. This makes it very easy to use newsgroup and world wide web resources in later courses.

5. Don't Forget the Classics!

As educators we are transmitting a culture to our students, the culture of computer science. One part of that culture is awareness of the "classical" literature of the field. One experience that we did not mention above is the experience of doing some background reading in computer science. Certain books have become so distinguished that they are part of the culture of computer science just as Shakespeare is part of the culture of English literature. All computer science students should have the experience of reading them. Fortunately, even beginning students will find these books quite understandable as well as quite interesting.

Computer science instructors have an obligation to prepare an appropriate list of these classics, make sure their students have the list, and encourage students to read these books. We have included below a short list of some of our favorites, with apologies for what are no doubt notable omissions.

ACM Turing Award Lectures, The First Twenty Years 1966-1985, ACM Press, New York, 1987.

Human Interface Guidelines: The Apple Desktop Interface, Addison-Wesley, Reading, MA, 1987.

Abelson, H. and Sussman, G. J. with Sussman, J. *Structure and Interpretation of Computer Programs*, The MIT Press, Cambridge, MA, 1985.

Bentley, J. *Programming Pearls*, Addison-Wesley, Reading, MA, 1986.

Bentley, J. *More Programming Pearls*, Addison-Wesley, Reading, MA, 1988.

Brooks, F. P., Jr. *The Mythical Man-Month*, (Anniversary Edition), Addison-Wesley, Reading, MA, 1995.

Dijkstra, E. W. *A Discipline of Programming*, Prentice-Hall, Englewood Cliffs, NJ, 1976.

Dewdney, A. K. *The Turing Omnibus*, Computer Science Press, Rockville, MD, 1989.

Goldstine, H. H. *The Computer from Pascal to von Neumann*, Princeton University Press, Princeton, NJ, 1972.

Gries, D. *The Science of Programming*, Springer-Verlag, New York, 1981.

Kernighan, B. W. and Plauger, P. J. *Software Tools*, Addison-Wesley, Reading, MA, 1976.

Neumann, P. G. *Computer Related Risks*, The ACM Press, New York, 1995.

Polya, G. *How to Solve It*, Doubleday & Company, New York, 1957.

Randell, B. (ed.) *The Origins of Digital Computers: Selected Papers*, Springer-Verlag, New York, 1973.

Weinberg, G. M. *The Psychology of Computer Programming*, Van Nostrand Reinhold, New York, 1971.

Wirth, N. *Algorithms + Data Structures = Programs*, Prentice-Hall, Englewood Cliffs, NJ, 1976.

6. Conclusion

Again the purpose of this paper, and our proposed list of experiences, is to stimulate thinking and promote further discussion. We hope that other faculty will examine this list, modify it, add to it, and share their thoughts. We hope they will judge what best serves the goals of their own students, and will then map any missing experiences into appropriate places in their curricula. And we hope to begin to encourage an environment where providing and evaluating experiences is considered a major activity and responsibility of the faculty.

7. Bibliography

1. Curriculum 68, *Commun. ACM 11*, 3 (Mar. 1968), 151-197.
2. Curriculum 78, *Commum. ACM 22*, 3 (Mar. 1979) 147-166.
3. Gibbs, N. E., and Tucker, A. B. Model Curriculum for a liberal arts degree in computer science, *Commum. ACM 29*, 3 (Mar. 1986), 202-210.
4. Computing Curricula 1991, *Commun. ACM 34*, 6 (Jun. 1991), 68-84.