# Teaching Problem Solving Techniques and Software Engineering Concepts Before Programming

**Rob Faux**
**Computing Department**
**United States Open University**
rfaux@oneota.net

**Abstract:**

This paper outlines research currently underway that seeks to determine the impact of teaching various concepts before a programming language. Many educators have espoused the concept of preparing learners for success in Computer and Information Science by teaching problem solving techniques, approaches to design and software engineering concepts prior to actual coding. While various efforts to implement this approach exist, very little empirical data has been accumulated. Course content effectiveness research in this area is relatively sparse.. This research measures the learning of two groups in a first programming course after participants complete pre-programming courses with varying content.

**Keywords**: introductory computer science, problem solving, CS0, CS1, CS curriculum, software engineering

## 1. Issues in Introductory Computer Science Courses

The gifted Information Systems (IS) or Computer Science (CS) student will succeed in nearly any permutation of pedagogy, curriculum and environment. The same can not be said of the majority of learners. Since IS/CS students must successfully navigate the Introductory CS courses in order to continue studies, it is appropriate that educators work to optimize course content and methodologies to aid the learning process. Improved instructional methods and curriculum content should lead to successful learning by a majority of the participants and will help many to pass through the gateway to these fields.

### The Introductory Course Debate

The introductory CS course has received considerable attention as educators attempt to identify how this course can be an effective gateway to the various IT professions. There has been considerable debate involving the languages, the tools, and the concepts to be included. Dale goes so far as to state that educators no longer know what to include in these courses as they once did [1]. At present, there are a number of educators and authors who espouse the idea of including non-programming topics either before programming or early in the first programming course [2]. In some cases, programs have instituted what are termed CS 0 courses. These courses attempt to teach the student concepts that are thought to be useful for future success in programming and other CS/IS studies.

Adding another course or additional material to the CS/IS program pipeline brings about a number of issues. First and foremost is the complaint that the curriculum is already crowded [3, 4]. The question here is whether the materials added at the beginning of the process do, in fact, prepare students for continued learning. If they do not, then it is possible that they only serve to clutter the curriculum. If they do, is it possible that crowding pressures in other courses will be alleviated by the value added to introductory courses? These arguments tend to center around the academic perception of these issues. However, CS/IS educators are becoming increasingly pressured to meet consumer needs as perceived by the students and their future employers [5]. Thus, the addition of materials early in the curriculum must also be salable in order to be effective. There is a need to uncover evidence to support the idea that this investment of time is reasonable and appropriate.

### 1.1 The Struggle

Learners entering the first programming course, which is most often the introductory CS/IS course, fight three particular battles as they work their way through the course: culture shock, problem solving, and syntax. Students entering any introductory course are bound to be largely unaware of the language, methods and style of the field. For that reason alone, introductory courses can be very difficult for the learner as they reacculturate themselves to the new environment [6]. CS/IS requires intense and repetitive use of problem solving skills, an area in which many persons lack training. Learners entering the field are being asked to pick up these skills with little guidance and support as they progress through undergraduate CS/IS programs. Almost to add insult to injury, learners must also learn a new language. Regardless of the choice (Java, C++, Pascal, Ada, etc), there is a syntax that frequently becomes the

paramount issue in the learners mind [21]. Frequently a learner will worry about a compiler error when they should instead be looking at a logic error in their solution design.

### 1.2 Aiding in the Struggle

One of the first steps in attempting to help learners to succeed in the introductory course is to provide a bridge from the student's current culture to the new culture the learner is hoping to join [6]. Providing concepts within a frame of reference that falls within the learner's experience is an important component of this process. Problems and concepts presented outside of the realm of the programming language should increase the likelihood that learners can be reached with the concept. Providing time for the learner to learn about the field and some of its vocabulary prepares them for their journey into the new culture. An example of this is provided by Vandenberg and Wollowski as they emphasize building a framework that gives the learner background for the things they will learn as they go through a CS/IS program [7].

Time should be invested in either honing or developing problem solving skills at the beginning of the curriculum. Many authors and instructors admit that problem solving skills are vital for success in CS/IS fields [9, 10, 11]. While many learners will develop these skills over time whether they are directed or not, the experience of many educators indicates that better problem solving methods would make the job easier for both students and instructors throughout the curriculum. Guided efforts in identifying problem solving practices and techniques could prove to be valuable to beginning learners.

The syntax struggle may be alleviated by the reduction of stress produced by culture shock and/or weaknesses in problem solving techniques. Recognizing the importance of these issues and providing time and guidance to address them gives the student more resources as they attempt to learn to program. This is contrary to programs where programming courses serve as the introduction to the major, the language takes precedence over solving the problem or the concepts being taught. Intuitively, language precedence seems counter-productive to future learning.

## 2. Existing Research

It is hoped that this study can provide some substantial support to the contentions that teaching certain topics early in the CS/IS curriculum can benefit the learner by providing them with tools that lead to future success in the field. While there is a great deal of literature that supports these ideas, very little of the literature takes the form of an empirical study. In most cases, the literature provides anecdotal or observational support. Those studies that do exist tend not to focus on curriculum content, but focus more on learning styles and teaching styles.

### 2.1 Learner Preferences and Background

The learner's background and personal preferences certainly have an impact on the ability to acquire knowledge. A great deal of general educational research exists regarding these issues. Studies that are specific to introductory CS/IS courses are less frequent. In some cases, the publication is based on the observation or experience of an instructor as they try new approaches. For example, Sanders and Mueller note that there is an increasing gap between students who have experience programming before college and those who do not [14]. This observation leads Roberts to suggest that providing multiple introductory course tracks is beneficial [15]. While these ideas have great merit, no studies, to my knowledge, back these observations.

An interesting study by Goold and Rimmer found that gender has an effect in first term programming, but that this tends to disappear in the second term [16]. It is possible that those females that were having difficulty chose not to go on in the program, however. This study also links the 'dislike of coding' to difficulties in the first two term coding courses, which may lend some insight to the possible need for providing a bigger picture to learners early in the process. Chamillard and Karolick confirm that learning styles do have impact on learning in CS1 courses, but also show that it is possible to help overcome learning style differences by preparing instructors to discuss different study methods with learners [17].

Finally, a study that links learner preferences with teaching techniques by Gibbs shows that learner independence and learner dependence have no impact on learning when the constructivist approach to learning is used [18]. This is interesting since this article cites other studies that show impact in more traditional approaches to teaching introductory courses. This would lend some credence to including the topics being measured by this study, since they do tend to support a constructivist viewpoint, though the application is not of that method.

### 2.2 Instructor Methods

The Gibbs study, and others cited by that work, investigate constructivist and other teaching pedagogies that indicate some impacts on learning. Similarly, collaboration in introductory courses has been shown to have some effects on learning at this level as well. Carter observes the varying levels of collaboration and attempts to ascertain where the line between collaboration and plagiarism may fall [8]. It is interesting to note the observed perceptions students have of certain actions as opposed to an instructor's view. A study that provides a good deal of reason for collaboration is the one provided by Chase and Okie [20]. This study found that the inclusion of peer instruction and cooperative learning in the introductory courses reduced the number of withdrawal, D grades and F grades (WDFs). Especially interesting here is the decrease in WDFs among female participants.

### 2.3 Curriculum Modifications

A large number of papers exist that outline the possible need for teaching various concepts prior to programming. In some cases, the writers espouse inclusion of various topics early in programming courses. In others, they advocate for a CS0 type course. In most cases, writers identify problem solving and structured design techniques as being important [21, 22]. Hilburn provides the idea that toolsets should be used in a top-down approach to teaching rather than teaching coding from the bottom up [23]. This method tends to imply abstraction and design methods to learning programming. Vandenberg and Wollowski go further and suggest that an early, breadth first course is important to set the basis for further learning [7]. However, not all papers support this approach. Buck and Stucki feel that early design is harmful and is contrary to Bloom's taxonomy [24]. While this is an interesting argument, this researcher disagrees with some of the interpretations of cognitive levels in programming outlined in this work.

Of particular note are studies by Sanders and Mueller [14] and Jackson and McCauley[19]. The first study provides support for the idea of teaching CS0-type courses prior to programming. The set of courses provided at the institution reported a significantly increased retention rate of learners through the early part of the program. It would be very interesting if further results could be determined for completion of the program. As the study stands at present, it does not conclusively show that the existence of these programs had lasting effects into the coding predominant courses. The second study shows that introduction to the basics of software engineering in the introductory courses led to better grades in advanced courses such as Operating Systems and Compilers. While this study provides some basis for optimism, the subjectivity of grading and the number of additional variables involved make it difficult to fully support the hypothesis.

## 3. Study Methods

The purpose of this study is to determine if the addition of particular topics to a pre-programming course has any effect on the learning of participants in the first programming course. Those topics selected are integrated into the existing curriculum of the participating institutions and include problem solving techniques, flow charts, algorithm testing, team development, and an introduction to software engineering and diagramming methods. For the purpose of continued discussion, the pre-programming course will be given CS0 as an identifier and the first programming course will be labeled CS1.

### 3.1 Overview of Study Environment

The Computer Science Departments at Bemidji and Mankato campuses of Minnesota State University have agreed to participate in this study. Both departments have implemented CS0 courses based on the text by Schneider and Gersting [12]. The course at Mankato is considerably newer than the one implemented at Bemidji. However, the existing course content was similar in terms of coverage in the Schneider/Gersting text. All instructors of CS0 have agreed to implement changes to their curriculum in the Fall, 2000 semester, which are provided by the researcher. A subset of instructors for the CS1 courses have agreed to allow data collection regarding learning in their courses during the fall and spring semesters of this academic year. All students attending the fall semester CS1 courses will not have had exposure to the modifications in the CS0 curriculum and will serve as the control group. Participants in CS1 courses during the spring semester will most likely have attended the CS0 course with the modifications during the fall. With the exceptions of those identified as having not taken the fall CS0 course, these participants will comprise the study group.

### 3.2 Modifications to the CS0 Curriculum

The existing CS0 curriculum is based off of a subset of the chapters provided in the Schneider/Gersting text. Material was selected with the pedagogical decision to avoid the introduction of a programming language in the course. Therefore, materials that relied on code were not considered. Further, any additional materials were required to follow this constraint. Existing topics in the CS0 course included algorithm design, algorithm efficiency, an introduction to hardware, computer systems organization, social issues and a brief introduction to the fields of computer science. Modifications introduced to the Fall, 2000 semester were integrated into existing materials. An introduction to problem

solving techniques is provided at the beginning of the algorithm discussions. Flow charting techniques and algorithm testing are provided in conjunction with algorithm development prior to algorithm efficiency. Team development issues are presented prior to the social issues discussions and the introduction to software engineering is included as a part of the field discussion.

### 3.3 Data Collection in the CS1 Course

The goal of data collection in the CS1 courses is to collect problem solving and programming data that will reflect the learning of participants in that course. The intent of data collection in this course is not to determine if materials from CS0 are retained. Instead, it is hoped that some significant difference may be discovered regarding the learning of materials provided in the CS1 course itself. In other words, the study is looking for the impact of the curriculum changes in CS0 on the learning of CS1 concepts.

Data will be collected on two fronts: demographics and demonstrated ability. The demographics provide an opportunity to collect sample population data that will help to verify the data collected as it relates to the general population. Further, the demographics collect information regarding participant opinions with respect to computing CS0, CS1 and IS/CS major programs of study. Demographic data will be collected at the beginning and the end of the CS1 course. All responses will be coded to allow for paired analysis. Responses are coded in a fashion that protects the anonymity of all respondents.

The ability of respondents will be measured at three points of time during the CS1 course. The first data measurement takes the form of a pretest. This pretest provides the participant with four problems that focus on different levels of problem solving typical to beginning programming. It is hoped that these pretests will provide a baseline ability level for the respondents. Once again, these pretests are coded for paired analysis. There will be additional data collected at the end of the course, when students are asked to complete a posttest. The posttest will include programming problems that reflect the same categories set forth in the pretest. The coding will allow the researcher to analyze the possibility that other factors, such as natural ability, are playing a role in the differences found in these events. In addition to this data collection, copies of individual programs completed by participants will be sent to the researcher for a project completed towards the end of the course. These projects will include selection and iteration structures and should illustrate method or function use and program organization. An article by Mengel and Yarramilli outlines some of the techniques the researcher will use to evaluate the programs [13].

### 3.4 Study Limitations

As with any empirical study, there are a number of limitations and factors that will influence the results of this study. Efforts have been taken regarding the design to reduce the impacts of these factors, but they should be noted for completeness. The sample size for this study is reasonably large, with an n that looks to be over 100 students per group (control and study). The relatively large size should reduce the impact of a number of extraneous variables and will provide further validity to any statistically significant finding.

Institutional and instructional differences will certainly have some impact on the successful learning of the respondents. It is certainly possible that Bemidji and Mankato will draw different populations of learners or that the environment of one place may have some advantage over the other. Further, the instructional capabilities of various instructors will likely introduce some variance in the quality of learning in these courses. Some of this is alleviated by the fact that only one instructor teaches the CS0 courses at each institution. Thus, there should be no variable in instructional quality between the two versions of the CS0 course being considered at each institution. There may be some difference between the two instructors at the different institutions. In the CS1 course, there are two different instructors at Mankato and a single instructor at Bemidji. Once again, the variation is minimal, but exists.

The issue of contamination of the control group is reduced since the introduction of the new materials is occurring during the same semester data is being collected for the control group. It is possible that some of these concepts could be shared by individuals currently attending CS0 with those currently attending CS1. However, it is unlikely that the CS1 participant will acquire all of the concepts covered in a way or time period that will greatly alter the results. Further, the larger sample size should mitigate this factor.

A topic of greater concern has to do with the quality of incorporation of the new topics by the participating instructors. While, the researcher has provided source materials, it is not possible to dictate the method of presentation throughout a course to the instructors. That sort of direction is counterproductive to learning and teaching and would provide its own set of biases. However, the impact on this study is that the researcher can not know the exact emphasis and coverage given to the new topics in these courses. The researcher can only know that these topics were included in a course, where previously they had not. Conversations with the participating instructors should provide further information regarding topic coverage.

## 4. Conclusion

The study outlined in this paper is intended to isolate the curriculum content prior to the programming segment of instruction in CS/IS programs. It is hoped that the results will give some indication as to the effect certain concepts might have on the learning of programming concepts in the first programming course. It is entirely possible that no significant difference will be found between the two populations. This, in itself, should not discourage the instruction of these topics. It is possible that there is more impact in later courses as students encounter more team projects or larger projects that challenge their intellect further. Many gifted students find the simpler programs in introductory courses to be easy and undeserving of preparation. However, many of these students may find a need for these tools as the problems get increasingly difficult. Regardless of the results, it is anticipated that this study will provide useful data for decision-making regarding introductory CS/IS courses in undergraduate programs.

**References:**


[1] Dale, N., "Possible Futures for CS2", *Proceedings of the 29th SIGCSE Technical Symposium on Computer Science Education*, Vol 30, No 1, Feb, 1998, p. 357.

[2] Marion, W., "CS1: What Should We Be Teaching", *SIGCSE Bulletin*, Vol 31, No 4, Dec, 1999, p. 35-8

[3]ACM/IEEE-CS Joint Curriculum Task Force Report (Summary). "Computing Curricula 1991", *Communications of the ACM*, Vol 34, No 6, June, 1991, p.68-84

[4]Proceedings of the 27th, 28th and 29th SIGSCE Technical Symposia on Computer Science Education, Vol 28, 29 and 30, Feb 1996, Mar 1997 and Feb 1998.

[5] Greening, T., "Pedagogically Sound Responses to Economic Rationalism", *SIGCSE Bulletin*, Vol 32, No 1, Mar, 2000, p.149-156.

[6] Bruffee, K, *Collaborative Learning: Higher Education, Interdependence, and the Authority of Knowledge*, The Johns Hopkins University Press, Baltimore, MD, 1993.

[7] Vandenberg, S. & Wollowski, M., "Introducing Computer Science Using a Breadth-First Approach and Functional Programming", *SIGCSE Bulletin*, Vol 32, No 1, Mar, 2000, p.180-4.

[8] Carter, J., "Collaboration or Plagiarism: What happens when students work together", *SIGCSE Bulletin*, Vol 31, No 3, Sep, 1999, p.52-5.

[9] Reed, D., Miller, C., & Braught, G., "Empirical Investigation Throughout the CS Curriculum", *SIGCSE Bulletin*, Vol 32, No 1, Mar, 2000, p.202-6.

[10] Chi, M., Bassock, M., Lewis, M., Reimann, P., & Glaser, R., "Self-explanations: How students study and learn examples in learning to solve problems," *Cognitive Science*, Vol 13, 1989, p 145-182.

[11] Sweller, J., ""Cognitive Load During Problem Solving: Effects on Learning," *Cognitive Science*, Vol 12, 1998, p 257-285.

[12] Schneider, G. & Gersting, J., An Invitation to Computer Science, 2nd Ed, Brooks-Cole, Pacific Grove, CA, 1999.

[13] Mengel, S. & Yerramilli, V., "A Case Study of the Static Analysis of the Quality of Novice Student Programs", The Proceedings of the 30th SIGCSE Technical Symposium on Computer Science Education, March, 1999, p78-82.

[14] Sanders, I., & Mueller, C., "A Fundamentals-based Curriculum for the First Year Computer Science", *SIGCSE Bulletin*, Vol 32, No 1, Mar, 2000, p.227-231.

[15] Roberts, E., "Strategies for Encouraging Individual Achievement in Introductory Computer Science Courses", *SIGCSE Bulletin*, Vol 32, No 1, Mar, 2000, p.295-9.

[16] Goold, A., & Rimmer, R., "Factors Affecting Performance in First-year Computing", *SIGCSE Bulletin*, Vol 32, No 2, June, 2000, p.39-43

[17] Chamillard, A., & Karolick, D., "Using Learning Style Data in an Introductory Computer Science Course", The Proceedings of the 30th SIGCSE Technical Symposium on Computer Science Education, March, 1999, p291-5.

[18] Gibbs, D., "The Effect of a Constructivist Learning Environment for Field-Dependent/Independent Students on Achievement in Introductory Computer Programming", *SIGCSE Bulletin*, Vol 32, No 1, Mar, 2000, p.207-11.

[19] McCauley, R., & Jackson, U., "Teachings Software Engineering Early – Experiences and Results", *SIGCSE Bulletin*, Vol 31, No 2, Jun, 1999, p.86-91.

[20]Chase, J., & Okie, E., "Combining Cooperative Learning and Peer Instruction in Introductory Computer Science", *SIGCSE Bulletin*, Vol 32, No 1, Mar, 2000, p.372-6.

[21] Proulx, V., "Programming Patterns and Design Patterns in the Introductory Computer Science Course", *SIGCSE Bulletin*, Vol 32, No 1, Mar, 2000, p.80-4.

[22] Gaona, A., "The Relevance of Design in CS1", *SIGCSE Bulletin*, Vol 32, No 2, June, 2000, p.53-5.

[23] Hilburn, T., "A Top-Down Approach to Teaching an Introductory Computer Science Course", Proceedings of the 24th SIGCSE technical symposium on Computer Science Education, 1993, p 58-62.

[24] Buck, D., & Stucki, D., "Design Early Considered Harmful: Graduated Exposure to Complexity and Structure Based on Levels of Cognitive Development", *SIGCSE Bulletin*, Vol 32, No 1, Mar, 2000, p.75-9.