On Teaching to the Masses and the Masters: Competing Requirements for Undergraduate Education

David Callele, Dwight Makaroff
Department of Computer Science
University of Saskatchewan
Saskatoon, Saskatchewan, Canada S7N 5C9
callele,makaroff@cs.usask.ca

ABSTRACT

Generational changes in the preparation of students for entering university have been substantial. The 21^{st} century high-school experience is very different from that of the prior generation. In particular, the social and academic skills developed seem to be those of reaction to external stimuli and peer-group conformance rather than individuality, personal responsibility, and problem-solving. Responsibility and activity tend to be concentrated in severely restricted environments, where the skills of creative problem-solving are not sufficiently emphasized and/or developed. The "Nintendo generation" of students tend to be visual learners [10] and they expect significant external motivation. However, introductory university courses in the sciences assume that independent curiosity motivates the expected problem-solving approach, a significant contradiction between expected and actual personality traits. The greater mass of the student body is ill-served by traditional pedagogy geared to those who readily master the materials.

In this paper, we present a strategy for introducing and reinforcing structured problem-solving strategies that are relevant to both the sciences and to the larger audience of the general student population. We include the motivation, teaching methods, and course topics. We stress active learning within a meaningful context that enables students to take the lessons from this course into the remainder of their undergraduate degrees and the rest of their lives.

Categories and Subject Descriptors

K.3.2 [**Computing Milieux**]: Computers and Education—*Computer and Information Science Education*

Keywords

Software engineering, requirements engineering, pedagogy, problem solving, solution strategies

WCCCE 2006, Calgary, Alberta, Canada

1. INTRODUCTION

Traditional university pedagogy assumes that, for a classroom of sufficient size, the demonstrated and measured capacity, skills, and preparedness of the students within the class can be approximated by a normal distribution. However, over the last 10 to 15 years, our experience has led us to believe that the current student body now demonstrates capabilities that tend toward a bimodal distribution (see Figure 1) when evaluated using similar techniques (assignments, laboratories, and examinations). This distribution has a lower, dominant, mode containing approximately 3/4 of the students, and peaks in the 35% to 55% range. An upper, subdominant, mode containing the rest of the student population peaks in the 75% to 85% range.

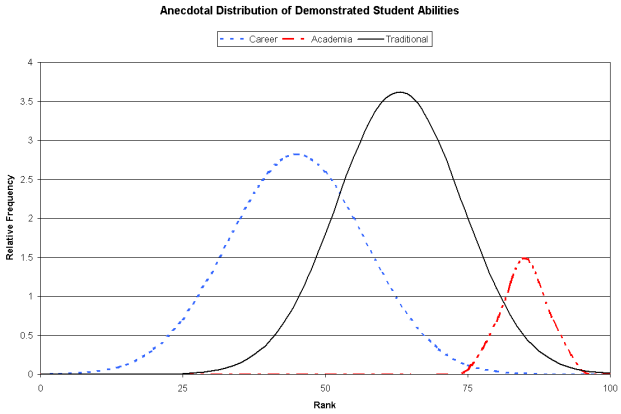

Figure 1: Anecdotal distribution of demonstrated student abilities

Our observations have generally been corroborated in discussions with our fellow faculty members in Computer Science from across North America and Europe. While there is some variation in the reported (bimodal) distribution, we do not feel that the differences in the precise values of the peaks in each mode are significant. The two modes are generally described by our peers as "those pursuing career training" and "those pursuing an academic career", although there are some in the career mode that desire to be in the academic mode. While these labels can be misleading, we shall make use of the designation *career mode* to refer to the group of students in the dominant mode and the designation *academic mode* to refer to the group of students in the

subdominant mode[1].

Other common elements from our discussions include the recognition that the student population is more diverse (by almost any measure) than ever before and that student emotional preparedness appears to be falling – most students demonstrate weaker "life skills" and skills commonly associated with a competitive environment.

These informal meetings with other faculty have led to many lively discussions on topics as varied as "What does the student body have a right to expect from us (their faculty) and from their curricula?" and "What are the goals of the university curricula? Should university computer science provide career specific education?" This discussion often leads to "Should we be teaching to the lower mode? The upper mode? Both? Do we have the resources available to do a proper job of teaching any of them?" Sometimes we even get the time to discuss "Should we try to help some, or all, of the students in the lower mode join the upper mode? Why?"

These questions are relatively abstract but they all derive from a common set of feelings among the practitioners:

> Too many of our students are not prepared for what we are teaching them and the way we are teaching them. Should we be changing the curricula to meet the students' demonstrated capacity? Or, should we be filtering the student body such that only those students that can handle the material are allowed to take the courses? Maybe we should be doing something completely different...

In this paper, we directly address this issue. After many hours of reflection and discourse with fellow educators, we believe that the introduction of a first year course in critical thinking, problem solving, and communication will better prepare the students for traditional university learning environments and help them move between the identified modes (if they so desire). An alternative to reducing the complexity of the curricula, or restricting the student body to the upper mode, our proposed curriculum for a first year course is useful to the student population at large.

Our analysis shows that the materials that we propose to gather in this new course are fundamental. These materials are "enablers for success", knowledge and skills that are critical to post-secondary learning. We advocate a proactive approach that makes explicit what is so often implicit, with emphasis on identifying problems, developing strategies for solving these problems, techniques for verifying that the solutions developed have indeed solved the problem and communicating with others about the entire process.

The remainder of this paper is organized as follows. Section 2 contains related work in recruitment, retention, and curriculum. In Section 3, we present anecdotal evidence and needs-based motivation for the specific issues addressed by our approach. Section 4 develops the principles upon which this course is based while Section 5 presents a course description, an outline of topics, and select activities. Finally, we conclude in Section 6.

[1]We have not attempted to perform a rigorous quantitative analysis across numerous student bodies given the current concerns over research ethics and student privacy; throughout this paper we report only anecdotal evidence.

2. RELATED WORK

Across North America and Europe, enrollment in Computer Science has been in steep decline over the last several years [6, 12, 15]). Despite predictions of a substantial shortage of IT workers by federal government agencies and various industry groups [1], the students have not returned.

One of the reasons postulated for declining enrollments is a fundamental misunderstanding of the nature of computer science and its relevance to problems facing society today, misunderstandings shared with many other disciplines [6]. For example, a common view among potential students is that *Computer Science* and *Programming* are the same thing; many students don't wish to spend their entire careers behind a keyboard with very little opportunity to deal with other human beings.

To this end, a number of approaches have been investigated and implemented (*e.g.* [2, 14]) to captivate students at an early stage, and to engage them with exciting application areas (*e.g.* healthcare, game development, bioinformatics) that they may encounter in their careers. In particular, Talton *et al.* [14] use a creative orientation session to assist in retention of first-year students.

The EECS department at the University of California at Berkeley has introduced a non-traditional first year course in Computer Science [7]. The course uses technology to solve HCI problems in a hands-on, project-oriented manner and claims that "the skills you develop, while not directly relevant to other computer science courses, will be useful wherever you go after Cal.". While our course does not focus so closely on HCI problems, our goal is similar: to provide those skills that will transcend the current classroom and the information presented therein.

Similar efforts at providing students with the skills to become better problem solvers and more emotionally mature have been made in the College of Education at the University of Saskatchewan [3, 9]. The course HLTH 100.3 (Health Concepts for Elementary and Middle Years) was taught during the Winter 2006 session, in an innovative, highly interactive manner that emphasized the application of the principles that we hold as critical: identifying problems, developing strategies for solving these problems, techniques for verifying that the solutions developed have indeed solved the problem and communicating with others about the entire process.

By the end of the term, after credibility with the instructor had been formed, students were challenging and questioning the course content in a meaningful and respectful way. Various active learning strategies were employed and there was significant engagement of the student body with corresponding success at applying the key principles. Students demonstrated teamwork and building a sense of community around shared problems (in the context of teaching health-related information in elementary school) and actively engaged in the practice of the principles: question everything, analyze the context of each piece of information you are given, and don't take anything for granted. Congruent with our own philosophy, the instructor described these outcomes as "enablers for success" at university and beyond.

In prior work [5], we presented a new class, CMPT 214 Programming Principles and Practices. CMPT 214 was introduced to our curricula in response to faculty perceptions of weaknesses in many students' abilities to cope with computer science course material. By the completion of their

undergraduate degree, many students didn't seem to really understand what a computer is, or what it can do, and had great difficulty mapping a solution from requirements to working software artifacts. In particular, one of the most troubling symptoms was the excessive time students were spending on their homework. Assignments that faculty expected to take eight to 10 hours were often taking in excess of 40 to 60 hours. When this was coupled with a just-in-time scheduling attitude toward assignments, both student and faculty frustration was significant.

After the first two years of CMPT 214, we have observed a marked difference in the student body. While some of the original goals of CMPT 214 were achieved, most particularly in the goal of teaching students to question their requirements and their assumptions about same, the overall effect upon the student body is less than we had hoped for.

Numerous reviews of CMPT 214 have been performed, including reviews by the faculty teaching the class, the students taking the class, reviews by other faculty, and reviews by the combined faculty-student curriculum committee. While the specifics of these reviews are the subject of another paper, for our current purposes the most relevant factor is an error in identifying student abilities at the second-year level. Despite significant efforts to the contrary, we still over estimated the skills and abilities of a typical second-year student.

3. MOTIVATION

In this section, we characterize the prototypical "early years" university student, followed by specific observations on computer science students. We then compare high school and university teaching environments and discuss perspectives on university teaching. Finally, we justify our positioning of this material in first year.

3.1 Characterizing Students

Intellectually, current students exhibit greater breadth of exposure to topics than 25 - 30 years ago. The top students are as good as, or better than, any other students within our experience. However, the number of students demonstrating weaker communication, critical analysis and problem solving skills is higher than ever before.

At the emotional level, we perceive our students to be strongly conditioned by their prior educational experiences in high school. There, the teacher is a purveyor of "truth", the (somewhat arbitrary) arbiter of right and wrong. Assignments and examinations are "perfectly stated" and there is no failure, except a failure to (minimally) try. However, as authority figures teachers have no real power and are not allowed to exert more than minimal pressure on the students. This lack of consequence for students can induce a cynical attitude that "this is all just a game" with no relevance, no sense of personal ownership of the material, and no sense of personal responsibility for their own success.

The examples in Table 1 are illustrative of the general mismatch between expected and actual behavior at the university level. A direct consequence of this mismatch, excessive effort on assignments, has already been mentioned. These incongruities can lead to a general emotional malaise among the students which can, in turn, lead to feelings of frustration and failure within the faculty.

Individuals in the career mode do not appear to be well served by the traditional university model. We have it within our power to help them make the transition part, or all, of the way to the academic mode. However, it will require determination and the skills of our best educators to achieve. This task is critical, for society needs the personnel that we are educating, and helping students make this transition should be given the importance that it deserves.

Table 1: Expected *vs.* actual student behavior

Expected	Actual
Review material before the lecture	Expect the lectures to cover *all* relevant material, no need to evaluate what is important
Start assignments immediately, schedule over time	Earliest deadline first, JIT scheduling
Proactive faculty interaction	Weak or non-existent classroom interaction, schedule appointments only after disaster strikes
Make a concerted effort to understand the intricacies of the issues	Expect all analysis to be completed before presentation, student responsible for following the outline and filling in the blanks

3.2 Computer Science Students

Within computer science, it appears that many incoming students with prior preparation in programming (such as a high school course or self-taught skills) tend to discount the importance of fundamental problem-solving techniques. They become over-confident due to their familiarity with syntax and development environments and then experience significant difficulty adapting to the more rigorous techniques required for addressing problems of increased complexity or difficulty.

In comparison, those students without programming experience become fixated on issues like programming language syntax and development tools, perceiving them as the focus of computer science as a discipline, rather than as a means to an end. This focus on the technology of programming distracts them, and for many, they can not or do not concentrate on developing the requisite problem-solving techniques.

In other words, students exhibit significant difficulty developing adequate familiarity and facility with both the tools and the abstractions needed for success at this level. We believe that these problems are not restricted to computer science students. Many otherwise capable natural and social scientists also find the transition to higher education difficult.

3.3 University *vs.* High-school

There are significant differences between the secondary and post-secondary learning experience. At the university level, there is some degree of absolute standard for evaluating students – particularly in those disciplines with active certification bodies. In general, students are required to show (communicate) that they have progressed (to varying degrees) through all levels of Bloom's taxonomy (knowledge, comprehension, application, analysis, synthesis, evaluation) [4] within a given course.

Unlike high school, failure *is* an option and assignments and examinations can be deliberately tricky. There is a very real power imbalance between the students and the faculty; a

student generally can't call upon their parents to "fix things" with a simple phone call. Students must also learn that not all evaluation is over a continuum, sometimes evaluation can be binary: either they get it or they don't and there are consequences if they don't keep up.

A typical student, living on their own for the first time, must also learn to take personal responsibility for (almost) everything in their lives. Not only must they care for all aspects of their personal well-being, they must also manage their time and ensure that they attend classes without any form of supervision (or even any indication that anybody cares). They must learn that effective teamwork is very important and that, while authority is not always right, it must be treated with respect. These are significant attitudinal changes that must occur, changes that appear to be ignored in current university structures.

University pedagogy focuses on content and intellectual maturation; emotional maturation is expected as a byproduct of 'time' and 'growing up'. However, is it reasonable to set goals for students that require specific levels of intellectual *and* emotional maturity while only providing formal instruction of the intellect?

Note the "byproduct" term in the prior paragraph. In general, there are many *implied* lessons and learning events in post-secondary education. Should they be made explicit? Do the students ever receive formal instruction the tasks and skills critical to success? At university? Anywhere?

Is this a problem with all students? With their high school education? With their families, or the way they were raised? We take the position that the underlying cause (or causes) are irrelevant; we can only deal with what we are presented and address those issues within our control. We apply a manpower model: we have new employees (first year students) and we need to apply some on-the-job training. What are the students missing, or what do they need to increase their probability of success, and how do we deliver the solution to them?

3.4 Another Perspective

If what we have written so far is true, then we need to decide whether to address this new reality or continue to teach in the manner we have always taught. We are intelligent individuals, we specialize in recognizing problems and solving them – why aren't we addressing this problem in a more proactive and visible manner?

We asked this question of numerous individuals at the SIGCSE 2006 conference. Typical responses were of the form:

- It isn't our job, we teach Computer Science, Engineering, Philosophy, etc.
- I don't have the training to do this, especially in an environment where active learning is as essential as this appears to be.
- We aren't their parents.
- I am afraid of the student response, feedback, or retaliation... Did you do this before or after you got tenure? I couldn't do it...

We believe that we must address these issues for it appears that the mandate of universities is changing. Once a bastion of free-thought and abstract knowledge, society now seems to expect that universities shall also provide career training, not just in the professional colleges but in all disciplines.

We can ignore this change at our peril, allowing the role of the university to be marginalized as funding cuts are applied and another institution rises to provide this career training. Or, we can embrace change, and determine how best to educate both the masters and the masses.

3.5 Where and When to Begin?

While a presentation of the details is beyond the scope of this work, we identified the materials for our proposed course by employing two analysis techniques. The first was a dependency analysis that investigated the prerequisite conditions for enabling success: If I want to teach topic X in 4^{th} year, then what do the students have to know before they start? What skills do I expect them to have? More generally, what are my assumptions about the students? The second analysis was holistic in nature, performing a top-down analysis of the field of computer science (and to some degree, engineering) in an attempt to identify the fundamental skills and techniques required for success.

Both of these analyses lead to approximately the same conclusions. We have captured the common materials, added connective material as necessary, and we present the result as the contents of our proposed first year course in the following sections.

4. PEDAGOGY

Our proposed class introduces creative, yet structured, problem solving techniques with special emphasis on techniques for ensuring comprehension of the problem statement. The course covers four principal topics:

1. Introduction to university education
2. Analyzing and understanding problems
3. Solving problems
4. Representing data and information

Material within these topic areas is applicable to the student body at large and is also addressed, in whole or in part, by many departments outside of the sciences.

We expect the class to be compulsory for science majors but accessible to students of all majors – there should be no content that requires specific domain knowledge beyond that required to set the context of the investigated problems, no technical sophistication that precludes non-science majors from participating. This exposure to other disciplines should expand student's appreciation of other topic areas and help them to make a more informed choice for their major. We also hope that this focus on cross-disciplinary problem contexts will motivate members of other departments to participate in the development and instruction of this class.

4.1 Motivation

This course provides formal instruction in those skills identified as critical for success at university. The critical analysis of problem statements, identification of assumptions, and techniques for obtaining missing information are all key skills. Communication skills, particularly as applied to communicating about reasoning and decision making, are necessary

if the student is going to convince their educator that they have some degree of mastery over course content. Problem solving techniques, like iteration and 'divide-and-conquer', can be presented as generally applicable solution patterns – independent of the syntax of a programming language. Finally, enabling the students to understand the difference between data and information and how to store and represent data to facilitate it's conversion to information provides them with a solid foundation for tackling large-scale problems or automating problem solving strategies via programming languages.

This may appear to be a 'retro' approach [13], reinstating general practices from the early days of computing. However, these practices remain part of many general engineering curricula to this day. While some of these techniques are covered in 2^{nd} year (and later) computer science courses, particularly those dealing with Software Engineering, by that time many students have adopted a much different approach to problem solving for their individual assignments. For some students, this may be too late and 'bad habits' are already developed which preclude the necessary analysis for the student to have a confident and holistic view of the problem. To these students, using good problem-solving approaches only seems relevant to 'Programming in the Large' as opposed to being an integral part of program development activity, no matter how small.

4.2 Teaching Methods

Active learning is essential in an introductory problem-solving course. Students learn by first observing experts apply the techniques then by applying the techniques themselves. A highly interactive learning environment is essential and the faculty must use appropriate classroom techniques to keep the students interested and motivated.

Teaching strategies must consider multi-modal student performance distributions – in maturity, ability and preparedness. For the course to be meaningful to all students, there must be varied levels of exercises and activities. Given that part of the course material is about abstraction, and the division of tasks into manageable units, certain exercises emphasize abstraction and planning without detailed execution of the plans. Other exercises begin with basic plans and the students are required to add detail to the plans to the level of fundamental operations.

The following list contains typical exercises that could be used within this course.

1. Introductory exploratory lectures to set the context for the students. Promote interaction with faculty as a necessity.
2. Active learning group exercises to familiarize the students with active learning techniques.
3. Written problem strategies and solutions exercises
4. Oral presentation of strategies. For example, one of the lessons could be to teach someone how to make a peanut butter sandwich.
5. Execution of other student's strategies. Closes the loop on the need to effectively communicate the solution strategy.
6. Introduction and reinforcement of the methods of critical thinking.
7. Analysis and Reasoning. Example solution methods used in multiple disciplines such as philosophy, music, humanities, and engineering.
8. Scientific method. The scientific method (observing and exploring phenomena, developing and testing a hypothesis, and generating conclusions) is a fundamental structure for experimental work. Iterative application to a problem will always yield results for even negative results provide new insight into the problem domain,
9. Engineering method. Art and science become engineering when a domain is sufficiently well understood that we can reliably solve it's problems. Engineering is about solving a problem within a constrained domain; for example, use only this given set of components to craft your solution. While the engineering method excels at crafting solutions within constraints, there are times when solving problems in this manner can lead to duplicating solutions from other domains. Therefore, the greater the set of components known and available, the less likely it is for re-invention to occur.

4.3 Student Activities

We must be very careful to create activities that are relevant and appropriate to students at all levels. While it is inevitable that the top students may find some of the material to be self-evident, they should learn that mastery of these techniques comes only with practice and that the ability to apply these techniques as a matter of reflex rather than of conscious deliberation provides them with distinct advantages. We must be careful to ensure that the top students remain interested and must continue to motivate them with challenges appropriate to their skill level.

Context is very important. While the sciences may be the obvious domains, critical thinking is used in all disciplines. By drawing upon other domains, we can construct individual contexts with direct relevance to the students. Motivating commerce students with business plans, biology students with ecological scenarios, and computer science students with video game analogies should be much more effective than a discourse on *modus ponens* and *modus tonens* – even though the same lessons are being taught.

5. CRITICAL ANALYSIS, PROBLEM SOLVING, AND COMMUNICATION

We now present further commentary on the four main topic areas identified in Section 4. We describe explicit orientation topics, problem and solution types, mapping between the problem domain and the solution domain, and finally, data representation issues. Many of these topics must be covered in parallel, so that students can be actively involved in solving problems, not just sitting and absorbing problem statement after problem statement.

5.1 Orientation and Context

The issues faced by incoming students were discussed at some length in Section 3. Many of the implicit 'keys to success' at university are made explicit during the first week of the course (see Table 2). One of the most obviously necessary (yet poorly achieved) skills is that of examination writing. Some students require as many as two to three years to figure out the pragmatic methods to use when writing

examinations. Rather than leave such an important topic to chance, we provide explicit instruction on heuristics for maximizing their examination scores; heuristics such as how to select which questions to answer and how to determine the order in which to attempt the questions.

Table 2: Orientation topics

Main topic	Subtopics
Introduction to university	What your professors expect of you What you can expect of your professors
Assignments, projects, examinations	Plagiarism and Group Work
The mathematics of marking	Who marks what? How much time do they have?
Doing assignments	Time management Estimating effort
Writing examinations	Value heuristics
What is important? What is not?	Rules for course content Will this be on the examination?
Decorum	Interacting with other students Interacting with faculty Classroom behavior Student code of conduct

5.2 Problem Solving

Students often feel overwhelmed by the sheer magnitude of the requirements represented within an average assignment. At times they don't even know where to begin. One of our goals is to show the students that, using well-structured techniques, problems can be decomposed as far as necessary, until they reach a problem that they *can* solve. Then, by integrating the solutions to the sub-problems they can compose the final solution. In this manner, we hope to provide students with the confidence and skills necessary for addressing problems at the university level.

Motivating the students in a Socratic manner, the evolution of their learning process is guided by the following questions. These aspects of problem-solving provide the basis for intellectual exploration and for student engagement in the problem solving process.

1. What kind of problems exist?
2. Are there patterns for solutions?
3. What problems can be solved precisely and accurately? What problems can be solved in a heuristic and/or pragmatic manner?
4. What parts of the solution are controllable? What parts of the problem are stochastic and what parts are unpredictable?
5. How can we restrict the problem domain so that the problem is solvable? Is this restriction reasonable or acceptable?

At a more specific level, the following list illustrates the components of a problem and each component's scope:

1. Recognizing the problem
2. Scoping the problem
3. Internal factors
4. External factors
5. Ability to restate the problem
   (a) In context
   (b) In plain language
   (c) Formally (conceptually), introducing the logical operators (implication, quantification, and/or, while)
   (d) Syntactic analysis, argument forms (from Philosophy)
   (e) Introduction to control structures

The first of these topics is by far the most difficult: Is thise a problem that can be solved, or is this something that is beyond the comprehension or ability of the students (or anyone else)? This distinction is difficult to convey, but the students must learn that there may be parts of otherwise intractable problems that *are* tractable. The problem solver must determine the resources that constrain the solution space, what is beyond their control, and what they can influence or control within the problem domain. In many cases, the constraints are both implicit and explicit (*e.g.* you must use binary trees and recursion to solve this search problem), but the implicit external factors can be difficult to recognize. One must then decide if sufficient resources are available and whether it is worthwhile to deploy them in solving the tractable subproblem(s).

Learning to identify tractable elements requires that the instructor clearly model numerous systems with significant student participation. Students can then be given independent exercises that give them practice in modeling, evaluating the relative complexity and cost of solving problems, and stress the students' ability to communicate what they have learned.

However, many students find it difficult to analyze and interpret word problems, particularly when attempting to symbolically represent the problem domain. We will show them simple mappings from English language constructs to symbolic representations using the following heuristics.

1. Nouns, clauses as objects
2. Verbs as actions or operators
3. Representation using elementary predicate calculus

Perhaps the ultimate expression of problem comprehension is the ability to state the problem in a formal, mathematical framework, complete with logical formalisms and provable characteristics. While this is a worthy goal, it is unrealistic for entering students (particularly students from the career mode, or other disciplines) to be able to accomplish in the early stages. By the end of the course, we expect that all students will be able to generate formal representations of problems. While these formalisms may be simple, they are a significant step toward more advanced techniques.

When problem solving, one or more elements of the following (partial) list of solution components is often employed. We explicitly identify these components, and their motivation, to the students.

1. Identify the domain. Once you know what domain you are working in, you can take advantage of prior work identifying its underlying principles.

2. Identify the goal. If you don't have a clear understanding of the goal, then you can't reach the goal in a deliberate manner.

3. Identify assumptions. Making assumptions explicit helps clarify the scope of the problem. It may also identify assumptions that invalidate the problem. Assumptions can exist in the mind of the problem presenter, in the problem statement, and in the mind of the problem solver. Unfortunately, clarifying the nature and existence of assumptions can be difficult.

4. Estimation. The scope of a problem can usually be further clarified by rough approximations. Fermi problems [8] are an example of this solution strategy. Estimates can be used to validate progress: if the estimated values are significantly different from those created by the problem solution, something may be wrong. Estimation is also useful to bound the effort required to solve the problem.

5. Asking questions. Asking appropriate questions is an indicator of maturity. Changing student attitudes from 'to ask questions is to fail' to 'to fail to ask questions is to fail' is critical.

6. Order of operations matters. Due care and attention to the process is important. Errors introduced early into the process can propagate and even compound. For example, when making a toasted peanut butter sandwich, it is important to toast the bread before spreading the peanut butter on it. If these steps are reversed, one could always make a grilled sandwich out of the toast, but that is solving a different problem.

7. Actions and consequences. Selecting among alternatives is often necessary. Identifying choices that are irreversible can be very important: exploring a potential solution that destroys the data is a high-risk decision. Are there alternatives that should be investigated first?

8. Iteration and step-wise refinement. Many problems can be shown solvable by a proof-of-concept that would not be acceptable as a final solution. Developing a prototype solution shows that critical issues have been addressed and that the remaining work is an issue of step-wise refinement. A convincing presentation that has a few details to work out shows that an appropriate methodology has been followed and is far better than presenting no solution at all.

9. Atomic elements, abstraction, and emergent behavior. Atomic elements are the fundamental building blocks within the solution domain. Abstraction hides their details and allows the student to generalize. Emergent behavior occurs when the atomic elements are connected as required by the solution. Together, these concepts embody the knowledge and processes necessary for the development of larger problem-solving constructs. As students learn to abstract and generalize, they begin to identify that there are patterns in the problems, the solutions, and in the emergent behavior.

10. Communication skills. If no-one learns of, or understands, your solution then the quality of your solution is irrelevant. Similarly, if you did not extract a clear problem definition from the individual with the problem, it is unlikely that your solution is relevant. When presenting a solution, success or failure can be highly dependent on the way the solution is presented. Careful selection of argument strategies and how to present trade offs often determines how well the solution is perceived, independent of its eventual implementation.

5.3 Patterns

There are recognizable patterns in both problems and their solutions. The problem-solving process requires some degree of creativity, but not as much as many students believe – it is a process that can be taught. However, students must accept that solving problems is more than just pattern recognition and watching other people do it. It involves practice, getting their 'hands dirty' and making mistakes. Then, when they see what works and what doesn't, they gain the ability to recognize, generalize, and address problems at a meta-level.

5.3.1 Problems

The following list of problem types is typical of nearly all of first and second year computer science. Unfortunately, in introductory computer science, the problem type is generally obscured by the overwhelming syntax of the programming language used to express a solution. We envision an explicit consideration of these problem types *without* resort to programming languages of any form: if a student can't solve the problem in their native language, then how can they learn a new language and solve it there? Again, active problem solving during in-class exercises is used to expose the students to the patterns within the problems so that they can learn to independently recognize the patterns.

1. Read-eval-print
2. Stimulus-process-response. This involves the iterative execution of the following operations: observe, collect data, process data, effect change, create artifacts
3. Cause-effect; causality
4. Linear problems; strict progression
5. Branching linear; directed acyclic graphs
6. Looping - subproblems with repetition
7. Recursion as a special alternative to looping

5.3.2 Solutions

The previous section described components of problem patterns that could also be considered solution patterns. We choose to keep the solution patterns separate because the solution patterns often involve recognizing the primitive control structures implicit within the problem. The separation also lays the foundation for the concept of a computer as a solution execution engine, an engine that solves types of problems, independent of the actual problem domain.

There are several methods introduced in computer science education to guide the solver to a maintainable and effective solution that can be communicated to another person. For example, solutions can be expressed in a system design document, or using UML diagrams, or even in the source code of

a programming language. In each case, the problem is considered solved but the solution is stated with ever-increasing precision.

We introduce solutions at a more abstract level than a system design document; we introduce them as patterns. The following list of solution patterns could be taken from any software engineering textbook. However, we believe that we must make these techniques fundamental tools that first year students learn as early as possible then employ throughout their careers.

1. Top-down
   (a) divide and conquer
   (b) iterative decomposition
   (c) recursive decomposition
2. Bottom-up
   (a) (very) constrained set of building blocks for the solution
   (b) incremental skill development. For example, learning to play a musical instrument or how to play golf.
3. Meeting in the middle: (practical) implementation leads to integration
4. Design patterns
   (a) do always
   (b) test and do (while)
   (c) iterate over known range (for)
   (d) iterate until test (do while)
   (e) recurse: this looks like what I just did, only smaller. Base case and recursive step. What are the data storage consequences?

5.4 Data and Information

Freshman and junior university students have only a weak understanding of the relationship between data and information. Many tasks that they face require the management and analysis of significantly greater amounts of data than they have ever before encountered. Table 3 summarizes the points we must make.

Table 3: Organizing principles

Principle	Implementation
Like near like	Databases, spreadsheets
Most important first	Caching, to-do lists
By purpose or need	Object-oriented design
Maintainability	Filing cabinet, binary trees
Fast retrieval	Indexing, hashing

When actually solving a problem, the data must be stored somewhere. Storage is also required for ongoing calculations, and for final results. We introduce elementary data structures to the students as models they can employ when solving their problems.

1. Lists, sets, manipulation operators
2. Boolean operators including union, difference, intersection
3. Order and equality
4. Survey of other elementary data structures

A motivating context could be the exchange problem – one consequence of performing an exchange is the need for temporary storage. A real-world example is "moving day in Montreal" [11]. Every year, almost everyone who is going to move that year, moves on July 1. So, where is the temporary space if people are exchanging residences? It is on the roadways, in the backs of the moving trucks. A series of interesting derivative problems can also be expressed: Can you estimate how many moving trucks are needed? If all moving must be completed in 8 hours, does the number of trucks change? If there were only 2 moving trucks in the entire city, how big would they have to be? How fast would they have to drive?

6. CONCLUSIONS

In this paper, we present the position that many current undergraduates are not prepared for traditional university education paradigms, particularly in computer science. This position is consistent with the views expressed by educators in other disciplines, such as education and engineering, about their own students. We have identified a need to explicitly teach problem-solving skills to the students, in an engaging and interactive manner, to improve their probability of success in university and in their careers.

We have designed a course for first-year students with emphasis on identifying problems, developing strategies for solving these problems, and communicating with others about the entire process. The course does not require previous exposure to computer programming and only a modicum of mathematical background. The problem-solving domains are those of every-day life and domains of specific student interest. While the methodologies presented to the students may be appropriate for software engineering courses, they are presented without the syntactic baggage of a programming language. Therefore, the class is accessible to students of all disciplines, not just those in computer science.

7. REFERENCES

[1] Omar El Akkad. Where jobs are and students aren't. http://www.theglobeandmail.com/servlet/ArticleNews/TPStory/LAC/20050921/CATEC21/TPBusiness/General, September 2005. Globe Technology.

[2] Casey Alt, Owen Astrachan, Jeffrey Forbes, Richard Lucic, and Susan Rodger. Social Networks Generate Interest in Computer Science. In *Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education*, pages 438–442, Houston, Texas, USA, March 2006. ACM Press.

[3] June Anonsen. Instructor, HLTH 100.3, private communication.

[4] B. S. Bloom. *Taxonomy of Educational Objectives, Handbook I: The Cognitive Domain*. David McKay Co Inc., New York, 1956.

[5] David Callele and Dwight Makaroff. Teaching Requirements Engineering to an Unsuspecting

Audience. In *Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education*, pages 433–437, Houston, Texas, USA, March 2006. ACM Press.

[6] Lori Carter. Why Students with an Apparent Aptitude for Computer Science Don't Choose to Major in Computer Science. In *Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education*, pages 27–31, Houston, Texas, USA, March 2006. ACM Press.

[7] Computer science 160. http://www-inst.eecs.berkeley.edu/ cs160/fa05/, September 2005. University of California at Berkeley: User Interface Design, Prototyping, and Evaluation.

[8] Fermi problem. http://en.wikipedia.org/wiki/Fermi problem, January 2006.

[9] Health 100.3. http://www.usask.ca/calendar/hlth/100-199/, January 2006. University of Saskatchewan: Health Concepts for Elementary and Middle Years.

[10] L. Layman, T. Cornwell, and L. Williams. Personality Types, Learning Styles, and an Agile Approach to Software Engineering. In *Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education*, pages 428–432, Houston, Texas, USA, March 2006. ACM Press.

[11] Montral's Moving Day Madness...! http://www.tourisme-montreal.org/Media%5FTarget/HotTopics/EN/HTML/181%5FEN.asp, 2006.

[12] David A. Patterson. Restoring the popularity of computer science. *Commun. ACM*, 48(9):25–28, 2005.

[13] G. Polya. *How To Solve It*. Princeton University Press, Princeton, NJ, 2nd edition, 1957.

[14] Jerry O. Talton, Daniel L. Peterson, Sam Kamin, Deborah Israel, and Jalal Al-Muhtadi. Scavenger Hunt: Computer science retention through orientation. In *Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education*, pages 443–447, Houston, Texas, USA, March 2006. ACM Press.

[15] J. Vegso. Interest in CS as a major drops among incoming freshmen. *Computing Research News*, 17(3), May 2005.