A Taxonomy For Computer Educators

Submitted

October 17, 2003

To the

Annual Conference

of

The Association of Computer Educators of Texas

“Issues on Emerging Technologies”

By

Robert P. L. Skip Ferguson
Table of Contents

Introduction ................................................................. 2
The Scientific Method ..................................................... 4
Logic ........................................................................... 6
Computer Science ....................................................... 8
Index .......................................................................... 11
Introduction

In the spring of 2003 the Texas Higher Education Coordinating Board approved the Computer Science Field of Study Curriculum. The first course in this Field of Study Curriculum is **COSC 1436 Programming Fundamentals I**. This course is described as follows:

**COSC 1436 Programming Fundamental I, Introduces the fundamental concepts of structured programming. Topics include software development methodology, data types, control structures, functions, arrays, files, and the mechanics of running, testing, and debugging. This course assumes computer literacy.**

As a working hypothesis I assumed that the new course was not just another programming language course. The Academic Course Guide Manual already lists an introduction to programming course with all of the modern languages and many of the old ones.

The first high-level programming languages Algol, FORTRAN and Lisp were available by 1959. Computer Science as a credentialed academic field had its beginning in 1963 at Purdue University. In 1984, Apple, with each Macintosh computer bundled the application HyperCard. This application brought programming to the masses. HyperCard, like the Mac was designed using the “Object” design principles of artificial intelligence.

The first Information Technology degree was accredited in 1988. Now virtually all post secondary schools have a CS department or an IT Department or both. Following the proliferations of the Intel PCs with MS Windows in 1995, many other college departments have added IT degrees. Schools of education, law and medicine have raised IT to a sub degree within their field. Spreading computer education over such a broad range has given rise to the problem of incompatible terminology, inconsistent product-to-product nomenclature or in the more scientific mode taxonomy for computer education.

In computer education there exists limited terminological agreement. The very name computing science has the alternatives information technology or information system. Fundamental terms like ‘state’ or ‘module’ or ‘specification’ or even ‘program’ may have different meanings in different contexts. Often such terms are not explicitly defined but simply borrowed from the language in use, or else are defined quite differently. It may very well be the case that only the term ‘bit’ qualifies for having a universally accepted meaning. A byte has not always been eight bits; the original meaning was a computer word. There are of course various computer word lengths, four, eight, sixteen, and thirty-two and thirty-six are the main ones.

This situation is an inevitable reflection of the fact that among educators there is no common understanding of the core of computer science—let alone a basic understanding among the postmodern public.

In this paper and presentation I will try to answer the questions: What is information technology; and what does computer science have to do with it.

---

1 Lower-Division Academic Course Guide Manual
2 Apple had been the number one selling personal computer in the world since 1979.
As an aid in this effort I will call on the work of Hanno Wupper of the Computing Science Institute Faculty of Mathematics and Informatics at the University of Nijmegen, Netherlands. Dr. Wupper has done extensive work in developing Formal Proofs. His *A Taxonomy for Computer Science* is frequently cited. Taxonomy for Computer Science proposes a consistent collection of clear and unambiguous terms for notions. These essential notions are of great consequences in the teaching of computer science and information technology.

Information technologists have the same principal goal as all technologists: to apply their science toward creating machines with certain properties and to prove these machines have the specified properties. The scientific method requires the application of First Order Logic to establish a proof. Some say programming is an art and a science, if so; Logic is the scientific part of algorithm design and implementation.

At this point the need for Formal Mathematical Methods is inescapable. The Proof language must be of the First Order Logic. Boolean Logic and the predicate calculus form the basis of First Order Logic.

Pay and job opportunities are two additional continuums by which we can compare the fields of computer education. The June 6, 2003, issue of the Houston Chronicle reported the entry-level salaries for CS graduates at $46,536 and for IT graduates at $39,800.3 Another economic measure that can be used is the number of L-1 visas issued each year for each field. L-1 visas are special work visas that are available to employers who can’t find enough qualified American citizens to fill their vacancies. The U.S. allows 65,000 special computer visas each year.

There are differences between computer science and information technology. Computer science uses the Scientific Method and has an established body of knowledge, which includes accepted procedures and practices. At this writing the professional societies concerned with the field of information technology (ACM, IEEE, AITP or AIS) are working on an agreed to body of knowledge. Two societies have a working straw man. Although there are many differences both fields share a need for scientific programming. From the computer science perspective programming is a tool for the investigation of computation (mathematical symbol manipulation). Information technology programming is the end in itself, that of building applications.

3 National Association of Colleges and Employers
The Scientific Method

Our postmodern education and culture has created a problem. There is a popular misconception that the scientific body of knowledge is determined by opinion polls, surveys, economics or magic. The postmodernist goes on to argue against logic and reason. These misconceptions increase the difficulty of teaching programming as a science. I accept that there exists the human Body of Knowledge, a set. This set is large and growing everyday but there exists a finite number of items in this set, and therefore a finite set. By the application of the scientific method we can determine which elements of this set of knowledge are in the "Science" sub-set and which parts are in the non-science sub-set. These two sub sets of knowledge are usually identified as the 'Arts' and the 'Sciences.' Science is a logical process and pure science is its own motivation and reward.

The Scientific method is the best way yet discovered for winnowing the truth from lies and delusion. The simple version looks like this:

1. Observe and measure some aspect of nature.
2. Invent a tentative description, called a hypothesis, which is consistent with what you observe.
3. Use the hypothesis to make predictions.
4. Test those predictions by experiments or further observations and modify the hypothesis in light of your results.
5. Repeat steps three and four until there are no discrepancies between theory and experiment and/or observations.

The following figure describes this process in a flow diagram.
Technology is an applied science. When a technology fails to follow its parent science strange things happen. Three examples illustrate the problems created when other influences outside of science guide technology: The Tucker automobile 1947; Sony Beta Max 1974; and Apple Macintosh Operating system 1985. In all three examples the superior technology’s implementation was altered by technologist following a non science hypothesis.

Each science has its own body of knowledge and its own set of procedures, e.g. physics, chemistry biology or computer science. The computer science body of knowledge includes; automated Formal System manipulation, Theory of computation, Formal Methods and the Lambda calculus. The major procedures of computer science are Recursive Evaluation and Functional Evaluation. The scientific method uses critical observation and precise measurements in all procedures. In the testing of a computer science hypothesis a schema is created as part of the formal proof. Science is an open society. All science is held up for public inspection. Science does not create; it discovers the laws of nature. All claims must be verifiable. Of all human institutions science may well be the only self correcting one.

Science is best defined as a careful, disciplined, logical search for knowledge about any and all aspects of the universe, obtained by examination of the best available evidence and always subject to correction and improvement upon discovery of better evidence. What's left is magic. And it doesn't work. -- James Randi
Logic

Logic is that branch of philosophy that deals with formal principles, methods and criteria of validity of inference, reasoning and knowledge. This section is a brief description of how the development of the electronic computer and logic are intertwined. Computer hardware is perfectly described or modeled using Boolean logic. Boolean logic is a Propositional logic. Propositional logic statements are either true or false. Since all "programs" can be imbedded in hardware all programs can also be perfectly described by Boolean logic. Mathematicians use Propositional logic as a proof technique. Gödel’s incompleteness theory does not invalidate this type of proof. Gödel’s theory requires a meta systems to form a proof. A program cannot prove itself correct. A Scheme built of First Order Logic (FOL), is a Meta system that can be used to prove the correctness of a program. FOL is created when Boolean logic is extended by Predicate Logic. The True false values of Boolean logic are manipulated with Boolean algebra; FOL is manipulated by the Predicate calculus using the quantifiers ∀, (for all) and ∃, (there exist) and discreet mathematics. The predicate calculus is a version of Alonzo Church’s lambda calculus. This is important because lambda calculus was proven in 1936 to be equivalent to Allan Turing’s Turing Machine. The Church/Turing theory states that any algorithm that can be computed can be computed by a Turing Machine or the equivalent solution via lambda calculus.

Boolean logic is the syntax and Predicate logic is the semantics of the mate-language that allows us to prove our specification => scheme => machine is correct. The mythical hardware/software interface is part of the machine; the program is part of the machine. The machine obeys all the laws of physics and follows the rules of logic. All mathematics is logical. That is it states what you can say and how to say it in such a manner that mathematics has meaning. Programming is the first level of abstraction in computer science. In order to retain the claim of science this abstraction must maintain a formal mathematical connection to the physical hardware.

Each branch of mathematics has it’s own syntax Addition Can be “infix” (4+3= ) or it can be “postfix” (+ 4 3). They both have the same Meaning or semantics, (7). Algebra, trigonometry and geometry each have their own syntax but Logic is always the semantics of mathematics.

A recent example of a logic failure was reported in the August 7, 2003, Houston Chronicle. The headline was “TAKS play Solomon, rules all answers right.” The subject of the headline is question 8 on this year’s tenth grade math TAKS test. According to the article the TEA intended the students to use the Pythagorean Theorem. This algorithm yields 36 cm as the answer. One teacher raised the concern that any student who has had Advanced Mathematics might use trigonometry and that would yield 27 cm as the answer. Both were given as possible answers. How can two answers be correct? Given in the stem of the question is the fact that this is a regular octagon. This tells the trig student that the angle of the octagon slice is 45 degrees. [Sin of 22.5 degrees multiplied by 4.6 cm should equal the square root of (4.6 + 4) cm.] It doesn’t because a regular octagon will not circumscribe a right triangle with a 4 cm side and a hypotenuse of 4.6 cm. The syntax of geometry and trigonometry were correct, the logic of the question is incorrect. Design logic errors are the most common errors in major programs.
Logic is the essence of rationality and the foundation of mathematics, on which in turn the whole of science and technology is based. Anon.

What is the perimeter to the nearest centimeter of the regular octagon drawn below?

F. 41 cm  
G. 36 cm  
H. 27 cm  
I. 18 cm
Computer Science

Computer science is an algorithmic science. Applied computer science is the study of a machine that is a combination of logic and electronics. The logic is implemented in electronics. The electronics form the substance of the computer. This substance has weight, occupies space and obeys all of the known laws of physics. Once the program is loaded in the computer’s primary memory (RAM) the program is part of the machine. Applied computer science emphasizes the creation of new technology. The fundamental questions of computer science are what can be computed and what problems have algorithmic solutions? The questions facing applied computer science are the following. What is the best computational approach to problems with algorithmic solutions? How can the discovery of algorithms be made easier? How can we communicate algorithms better?

Scientific programming is distinguished from other types of programming by the proof methods used. Formal methods provide the meta system that that allows us to develop proofs for our program.

People who think computer science is about computers probably think astronomy is about telescopes.
Formal Methods

A computer system may fail to perform as expected because a physical component fails or because a design error is uncovered. For a system to be both ultra-reliable and safe, both of these potential causes of failure must be handled.

Established techniques exist for handling physical component failure; these techniques use redundancy and voting. The reliability assessment problem in the presence of physical faults is based on Markov modeling techniques and is well understood.

The design error problem is much greater. A scientifically justifiable defense against this threat is seldom used in practice. There are 3 basic strategies that are advocated for dealing with the design error:

**Testing** (Lots of it)

**Design Diversity** (i.e. software fault-tolerance: N-version programming, recovery blocks, etc.)

**Fault Avoidance** (i.e. formal specification/verification, automatic program synthesis, reusable modules)

The problem with life testing is that in order to measure ultra reliability one must test for exorbitant amounts of time. For example to measure a $10^{-9}$ probability of failure for 1 hour one must test for more than 114,000 years.

Many advocate design diversity as a means to overcome the limitations of testing. The basic idea is to use separate design/implementation teams to produce multiple versions from the same specification. Then, non-exact threshold voters are used to mask the effect of a design error in one of the versions. The hope is that the design flaws will manifest errors independently or nearly so.

Traditional engineering disciplines rely heavily on mathematical models and calculation to make judgments about designs. For example, aeronautical engineers make extensive use of computational fluid dynamics to calculate and predict how particular airframe designs will behave in flight. We use the term ‘formal methods’ to refer to the variety of mathematical modeling techniques that are applicable to computer system (software and hardware) design. That is, formal methods is the applied mathematics of computer system engineering, and, when properly applied, can serve a role in computer system design analogous to the role it serves in aeronautical design.

Formal methods may be used to specify and model the behavior of a system and to mathematically verify that the system design and implementation satisfy system functional and safety properties. These specifications, models, and verifications may be done using a variety of techniques and with various degrees of rigor. The following is an imperfect, but useful, taxonomy of the degrees of rigor in formal methods:

**Level-1:**

Formal specification of all or part of the system.

**Level-2:**

Formal specification at two or more levels of abstraction and paper and pencil proofs that the detailed specification implies the more abstract specification.
**Level-3:**

Formal proofs checked by a mechanical theorem prover.

Level 1 represents the use of mathematical logic or a specification language that has a formal semantics to specify the system. This can be done at several levels of abstraction. For example, one level might enumerate the required abstract properties of the system, while another level describes an implementation that is algorithmic in style.

Level 2 formal methods goes beyond Level 1 by developing pencil-and-paper proofs that the more concrete levels logically imply the more abstract-property oriented levels. This is usually done in the manner illustrated below.

Level 3 is the most rigorous application of formal methods. Here one uses a semi-automatic theorem prover to make sure that all of the proofs are valid. The Level 3 process of convincing a mechanical prover is really a process of developing an argument for an ultimate skeptic who must be shown every detail.

Formal methods is not an all-or-nothing approach. The application of formal methods to the most critical portions of a system is a pragmatic and useful strategy. Although a complete formal verification of a large complex system is costly, but a great increase in confidence in the system is obtained by the use of formal methods.

The Following diagram illustrates the corner stones of formal methods developed by Dr. Wupper.
### Index

<table>
<thead>
<tr>
<th>Term</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>algorithm</td>
<td>3, 6</td>
</tr>
<tr>
<td>Boolean</td>
<td>3, 6</td>
</tr>
<tr>
<td>Formal</td>
<td>3, 5, 9</td>
</tr>
<tr>
<td>Lambda calculus</td>
<td>5</td>
</tr>
<tr>
<td>Logic</td>
<td>3, 6, 8</td>
</tr>
<tr>
<td>machines</td>
<td>3</td>
</tr>
<tr>
<td>module</td>
<td>2</td>
</tr>
<tr>
<td>postmodern</td>
<td>2, 4</td>
</tr>
<tr>
<td>predicate calculus</td>
<td>3</td>
</tr>
<tr>
<td>program</td>
<td>2</td>
</tr>
<tr>
<td>programming</td>
<td>2, 3, 4</td>
</tr>
<tr>
<td>prove</td>
<td>3, 6</td>
</tr>
<tr>
<td>set</td>
<td>4, 5</td>
</tr>
<tr>
<td>specification</td>
<td>2</td>
</tr>
<tr>
<td>state</td>
<td>2</td>
</tr>
<tr>
<td>Technology</td>
<td>5</td>
</tr>
</tbody>
</table>