WHAT IS COMPUTER SCIENCE ALL ABOUT?

H. Conrad Cunningham and Pallavi Tadepalli
Department of Computer and Information Science
University of Mississippi

COMPUTERS EVERYWHERE

As scientific and engineering disciplines go, computer science is still quite young. Although the mathematical roots of computer science go back more than a thousand years, it is only with the invention of the programmable electronic digital computer during the World War II era of the 1930s and 1940s that the modern discipline of computer science began to take shape. As it has developed, computer science includes theoretical studies, experimental methods, and engineering design all in one discipline.

One of the first computers was the ENIAC (Electronic Numerical Integrator and Computer), developed in the mid-1940s at the University of Pennsylvania. When construction was completed in 1946, the ENIAC cost about $500,000. In today's terms, that is about $5,000,000. It weighed 30 tons, occupied as much space as a small house, and consumed 160 kilowatts of electric power. Figure 1 is a classic U.S. Army photograph of the ENIAC. The ENIAC and most other computers of that era were designed for military purposes, such as calculating firing tables for artillery. As a result, many observers viewed the market for such devices to be quite small. The observers were wrong!

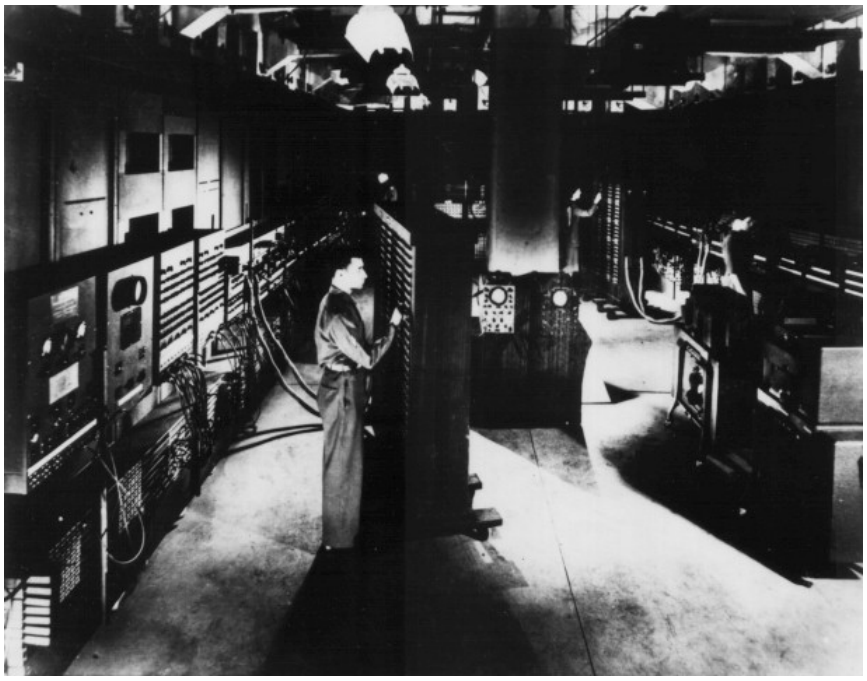

Figure 1. ENIAC in Classic U.S. Army Photograph

Electronics technology has improved greatly in 60 years. Today, a computer with the capacity of the ENIAC would be smaller than a coin from our pockets, would consume little power, and cost just a few dollars on the mass market. Figure 2 shows the Intel Pentium M microprocessor. This processor, which was designed for mobile devices such as laptop computers, is smaller than a dime, draws just 21 watts of power, and costs just a few hundred dollars. Figure 3 shows a handheld computer (i.e., a personal digital assistant, PDA), a far cry from the days of the ENIAC.

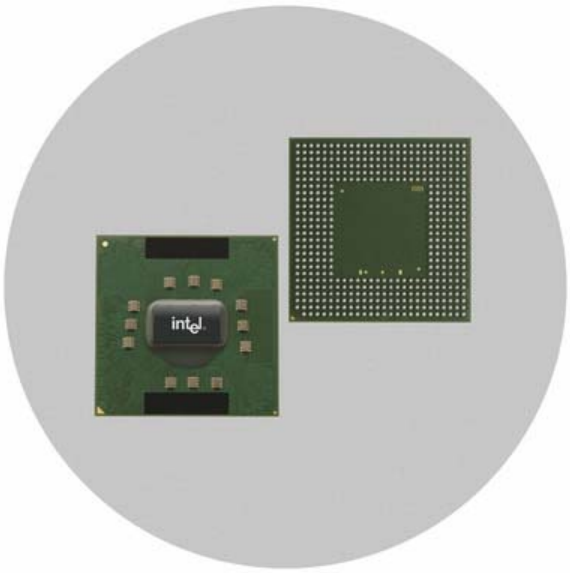

Figure 2. Intel Pentium M Microprocessor
[Intel Publicity Photo]

Figure 3. PDA
[Microsoft Word Clip Art]

Computers are everywhere. They are, of course, in banks, offices, and dorm rooms. We use desktop and laptop computers to send and read electronic mail, surf the web, and play games. We see the dramatic results of the use of computers in the exciting special effects in movies. But computers are also in less obvious places. They are what make our digital

watches, our cellular phones, and our MP3 players work (e.g., Figure 4). They are in the cars we drive, the gasoline pumps from which we fuel our cars, and in the traffic signals that steer us safely as we travel about. They are also in the rockets that are launched into space, as shown in Figure 5.

Figure 4. Computers Driving an MP3 Player [Microsoft Word Clip Art]

Figure 5. Computers in Action in a Rocket [Copyright free, http://gimp-savvy.com]

The computer is an enabling technology. It can be combined with preexisting technologies to enable some combination of increased functionality, decreased costs, and increased design flexibility. The computer can enable new technologies that might not have been possible otherwise.

How does a computer accomplish this? It is a device that controls some *process,* that is, a series of actions to achieve some desired result. It does so by encoding that process internally. The process is thus relatively easy to change as the desired result evolves.

Consider the example of traffic signals given above. The process of interest is the safe and efficient movement of many vehicles through an intersection (e.g., Figure 6). Because of the driving rules associated with STOP signs, the installation of such signs is an inexpensive solution for low-volume intersections. Unfortunately, STOP signs are not very efficient or safe as the traffic volume increases. The replacement of the STOP signs by traffic lights accommodates higher volumes more efficiently by routing several vehicles at once through the intersection. The traffic signal controls the process of traffic movement more explicitly than STOP signs do. However, the early traffic lights did not work well when the traffic volumes were irregular, changing from minute to minute, season to season. The installation of an adaptive traffic signal system can improve the situation. It can detect changes in the traffic patterns and adjust the signal times to enable more efficient movement of traffic through the intersection. The computer is a technology that enables an adaptive traffic signal to be designed and implemented in an effective manner.

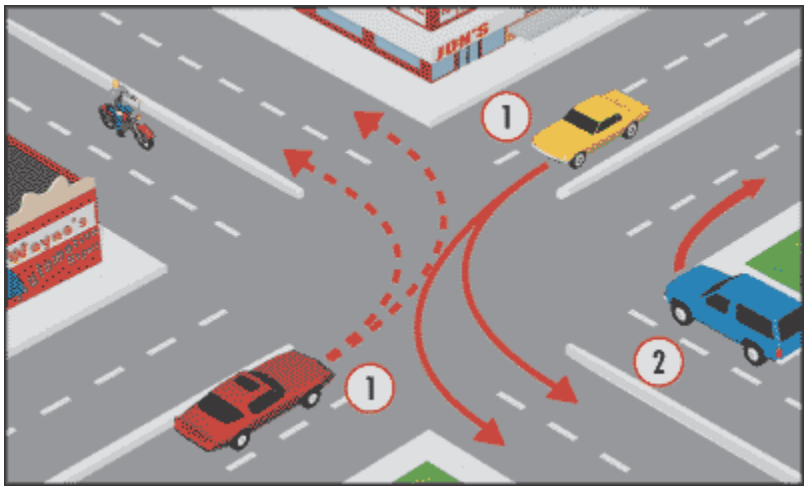

Figure 6. Turns at Intersection on Multilane Streets
[http://www.dmv.ca.gov/pubs/hdbk/pgs25thru29.htm#turnexamples]

The professionals who design computers and apply them in ever-expanding ways are the computer scientists, computer engineers, and information systems technologists. In some sense, we might describe *computer scientists* as the professionals who *put a process into the computer* and the *computer engineers* as the professionals who *put the computer into a process*. That is, computer engineers design the computers and integrate them into systems that can control a process of interest. It is the computer scientists who analyze the process of interest and describe how the process is to be designed and implemented as computer software. Similarly, it is the *information systems technologists* who apply the tools designed by the computer engineers and computer scientists to the needs of an organization, such as a business. The purpose of this article is to give you a hint of how computer scientists think and what they do.

THE SCIENCE OF COMPUTING

At its core, the science of computing involves the rigorous study of *process*, in particular, the study of processes that can be carried out by the devices we call computers. The scientific (and practical) questions of interest include: How do we carry out a process methodically? How do we describe what we do so that computers can carry out the process? How do we describe the objects that we are processing? How do know that we are doing it correctly? How do we do it efficiently? Are there some things we cannot do efficiently? Are there some things we cannot do at all? How do we carry out the process if we can do many things at the same time? How do we organize our work to describe a process, especially as the process becomes large and complex? And so forth.

Consider a different example, the process of baking chocolate chip cookies. If a person wishes to bake a batch of chocolate chip cookies (at least for the first time), he or she consults a *recipe*, such as the famous recipe for Toll House cookies shown in Figure 7 [http://www.VeryEasyBaking.com]. The recipe assumes that there is a human baker and a kitchen available that is equipped with an oven and appropriate baking utensils. The

recipe also assumes that the needed ingredients have been collected. The recipe then describes exactly what needs to happen for the baker to take the ingredients and use the equipment available to bake the cookies. The recipe gives a sequence of instructions that are meaningful to a baker in the kitchen: preheat, combine, beat, add, stir, drop, bake, and cool. The recipe also gives some indication of the amount of time it takes to bake the batch of cookies and how many cookies to expect.

Original Nestlé Toll House Chocolate Chip Cookies

Estimated Times:
Preparation - *15 min* | Cooking - *9 min* | Cooling Time - *15 min cooling* | Yields - *60*

Ingredients:

- 2 1/4 cups all-purpose flour
- 1 teaspoon baking soda
- 1 teaspoon salt
- 1 cup (2 sticks) butter or margarine, softened
- 3/4 cup granulated sugar
- 3/4 cup packed brown sugar
- 1 teaspoon vanilla extract
- 2 large eggs
- 2 cups (12-ounce package) NESTLÉ TOLL HOUSE Semi-Sweet Chocolate Morsels
- 1 cup chopped nuts

Directions:
PREHEAT oven to 375° F.

COMBINE flour, baking soda and salt in small bowl. Beat butter, granulated sugar, brown sugar and vanilla extract in large mixer bowl until creamy. Add eggs, one at a time, beating well after each addition. Gradually beat in flour mixture. Stir in morsels and nuts. Drop by rounded tablespoon onto ungreased baking sheets.

BAKE for 9 to 11 minutes or until golden brown. Cool on baking sheets for 2 minutes; remove to wire racks to cool completely.

Figure 7. Recipe for Nestlé Toll House Chocolate Chip Cookie
[http://www.VeryBestBaking.com]

In computer science terminology, the recipe for the process of baking chocolate chip cookies is called an *algorithm*. The *inputs* to the algorithm are the ingredients and the batch of cookies is the *output* of the algorithm. A *program* is the description of the recipe written in a specific language. This is the *software* for carrying out the process. The person who writes the program is called a *programmer*. The specialized language in which the program is written is the *programming language*. The *hardware* supporting the process consists of the oven, the baking utensils, and, of course, the baker. The process of applying the recipe to bake cookies is called *execution* of the program.

ENGINEERING OF COMPUTER PROGRAMS

Computers may have been called "giant brains" in old science fiction movies, but that is nonsense. At their basic level, computers are incredibly dumb. They need to be given very precise instructions with no room for ambiguity. The only language understood by a computer's hardware directly is its *machine language*. Each *instruction* in the machine language is just a sequence of 0's and 1's encoded as strings of *bits* (Figure 8) in the computer's memory. The data that these instructions act upon are themselves just strings of bits that encode numbers, letters, and other values. The instructions describe simple actions such as to add two integers or move data from one memory location to another.

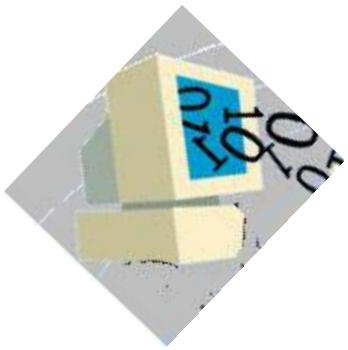

Figure 8. Computer and Its Bits
[Microsoft Word Clip Art]

A process that a human programmer wishes to encode within a computer is typically stated in much different terms than sequences of bits. How can we bridge the gap between the high-level concepts in the process as understood by humans and the low-level actions that can be carried out by a computer?

Let's look back at the recipe in Figure 7. It gives the direction to "stir in the morsels and nuts." Think about this for a moment. This is really a high-level description of a whole bunch of small actions such as: pick up 10 morsels and 3 nuts, put these in the mixing bowl, pick up a spatula, insert the spatula into the batter in the bowl, move spatula in a circular motion for 7 revolutions, remove the spatula (do not lick it!), put down the spatula, and then repeat this sequence again until all morsels and nuts are evenly distributed in the batter. Then, if we think about each of these smaller actions, we see that they are also high-level descriptions of even lower-level sequences of actions. And so forth.

Similarly, if we wish to design a computer program for a process, we start with a high-level, but precise, description of the process and its data (i.e., the *requirements*) and *decompose* each action or type of data into simpler actions and data. We then focus on each simpler item and decompose it further. We continue this technique until we can express the actions and data in terms that are meaningful to the computer hardware. We often organize the resulting software design into a *layered architecture* as show in Figure 9. One layer provides small processes that can be used to implement the larger processes

at the level above it. The layer, in turn, is implemented in terms of the processes in the layer below. We can think of a layer as a *virtual machine* of software instructions available to implement programs in the higher layer.

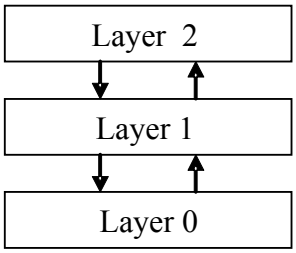

Figure 9. Layered Architecture

Of course, programmers don't have to go all the way to the machine language layer every time. They can *reuse* a lower layer that has been designed and implemented previously. They can also use specialized *software tools* (i.e., programs) to help develop the desired new program. For example, consider Figure 9 again. Suppose Layer 1 represents a high-level programming language. A programmer might then use that programming language to write a new program in Layer 2. Then the programmer can use a software tool called a *compiler* to translate this high-level language program to a machine language program that is supported by the machine (Layer 0). This is illustrated by Figure 10.

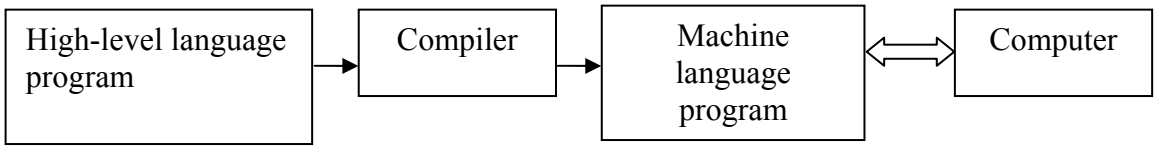

Figure 10. Translating between Layers

Although computers are superb at handling an enormous number of details at one time, human beings are overwhelmed when the number and complexity of the details get beyond the capacity of our finite minds. To be productive, we must simplify the situation. A layered architecture, as described above, illustrates a more general approach to dealing with complexity in computer programs—abstraction.

Abstraction means to ignore the unimportant details and focus on the essential features. We must look for common patterns and represent groups of low-level entities and actions as higher level entities and actions. For example, a novice baker may need to refer the detailed recipe for chocolate chip cookies given in Figure 7. A more experienced apprentice baker may be comfortable with a (more abstract) sequence of higher level instructions such as "combine ingredients into batter and drop on ungreased baking sheet." A master baker might only need abstract instructions such as "bake a batch of chocolate chip cookies" and "bake an apple pie." The abstractions enable the baker to handle many complex processes that must be done in conjunction with each other.

There are two kinds of abstractions at work here. *Procedural abstractions* are abstractions of actions, for example, "bake." *Data abstractions* are abstractions of objects that are acted upon, for example, "a batch of chocolate chip cookies." Both kinds of abstraction are important in the engineering of computer programs.

THE PROFESSIONAL DISCIPLINE OF COMPUTER SCIENCE

Computer science is a scientific and engineering discipline. Although computer scientists may play computer games, surf the web, use word processors and spreadsheets, and even open up the case on a PC and tinker with the hardware from time to time, those are not activities that describe what it means to be a computer scientist. Like other scientific and engineering disciplines, computer science involves a body of knowledge (i.e., a set of facts and theories), a set of practices that guide work in the field, and (perhaps more importantly) a way of thinking. We described some aspects of the way of thinking in the previous sections—understanding processes and the engineering of a layered architecture of appropriate abstractions to encode the processes for execution by a computer.

What computer scientists do can be characterized in two ways. First, there are the purely scientific activities noted in a previous section—the study of processes and how processes can be represented as algorithms and programs. Computer science researchers work to extend and deepen our understanding of these fundamentals. Second, there are the more practical activities of using an understanding of the science of computing to design and implement algorithms and programs to solve real-world problems using computers. The real-world problem might be an automated bakery, an adaptive traffic signal, a chess-playing computer, the special effects for the next blockbuster movie, or the operation of a global system to detect earthquakes and tsunamis.

The various scientific and practical activities have their bases in the different topics studied by computer science students. Well-educated computer scientists must have an understanding of what processes can be represented as algorithms for a computer and what cannot. This topic is called the *theory of computation*. They should also be able to determine how efficient an algorithm is, that is, how much time and storage space does the algorithm take to run in proportion to the size of its inputs. This topic is known as the *analysis of algorithms*. To keep track of the data needed for algorithms to work efficiently, a computer scientist must organize that data in an appropriate manner within the computer. This topic is known as the study of *data structures*. If the set of data is diverse, huge, long-lived, or used by several different processes, then a *database* may be needed to organize and store the data. That is another major topic of study for computer science students.

As a practical matter, computer scientists must have a mastery of the fundamental methods and technologies of computing. They must be able to develop an algorithm for a problem and describe it in a representation appropriate to execute on a computer. This basic task is called *programming* and the notation used for describing programs is a *programming language*. The programmer should verify that the program correctly solves the problem and test it to discover any defects. Program development and execution depend upon other technologies. To use computers effectively, programmers must

understand the basics of the *computer organization* and *network architectures.* They must also have a working knowledge of key system-level software technologies. These include the *operating system* that manages and controls a computer's basic operation, *networking* software that enables many computers to communicate and coordinate their activities, and programming language *compilers, interpreters, and runtime systems* that translate the programming notations to instructions the computer can execute directly.

Although computer science practitioners work with computers, they must also consider the human and social aspects of computing. While programming develops a computer program to solve a problem, *systems analysis* helps determine what problem needs to be solved to meet the human needs. While the theory of computation deals with what can be done with computers, the topic of *computing ethics* is concerned with what should (and should not!) be done from ethical, social, and legal perspectives. While analysis of algorithms and data structures deal with the effectiveness of the computer algorithms, the topic of *human-computer interaction* is concerned with the effectiveness of the human users of the computer program. While programming deals with the software development activities of a single human, the topic of *software engineering* is concerned with the situation where the software becomes too large and complex for one person to master. It deals with both the technical and the organization challenges of large software development projects.

Although computer hardware and software are interesting objects on their own, it is the application of computers to real-world problems that have the most practical significance. Many emerging applications of computers are areas of study and research by computer scientists in conjunction with other scientists, engineers, mathematicians, and subject matter specialists. For example, the field of *bioinformatics* looks at how to gather, store, process, and interpret huge sets of scientific data drawn from biological, biochemical, and medical experiments and tests. It seeks to use computing tools to understand life and develop cures for diseases. Another active area of application is *digital media* that looks at effective ways to gather, store, present, and interact with information consisting of a multimedia mix of text, two- and three-dimensional graphics, animation, sound, and touch. Digital media include such technologies as computer games, training simulators, and interactive Web sites.

PREPARING TO PRACTICE THE PROFESSION

How should a person prepare to practice this exciting and important discipline called computer science?

At the high school level, the preparation is almost the same as for any other scientific or engineering discipline. It is important to have a grasp of the fundamental pre-calculus mathematics (algebra, geometry, and trigonometry). Some study of calculus or discrete mathematics concepts would be helpful but is not essential. The ability to think logically and reason about quantitative concepts is more important than a lot of rote learning of facts or methods. It is important to understand the scientific method and have a good understanding of the basic principles of chemistry, physics, biology, and earth science. As should be the case with all well-educated citizens, it is also important to be able to

communicate clearly and effectively in both written and oral forms. It is important to have an understanding of human society that can come from the study of history, government, and other social studies. It is not essential to have previous computer programming experience, but it is helpful if you are comfortable using computers.

At the college level, you should study in a program in computer science. An ABET-accredited professional program in computer science typically includes:

- five or six courses in mathematics including differential and integral calculus, discrete mathematics, linear algebra, and probability and statistics
- four courses in the natural sciences
- two courses in English composition
- one course in speech
- seven other courses in the humanities, social sciences, and fine arts
- a number of courses in computer science including introductory concepts, programming in a high-level language such as Java or C++, data structures, computer organization, digital logic, operating systems and networks, theory of computation, analysis of algorithms, programming language organization, databases, software engineering, and professional ethics.

In addition to the formal coursework, it is important for you to obtain practical experiences in computing through internships, co-operative education experiences, part-time jobs, and general tinkering with the technologies being studied.

PRACTICING THE PROFESSION (OR ANOTHER)

The study of computer science at the college level is a good preparation for many fields of professional endeavor. It encourages the development of problem-solving skills, logical reasoning, attention to detail, and abstraction abilities. These serve the graduate well in his or her future studies, career, and life.

After study of computer science at the undergraduate level, some graduates will go on to graduate study in computer science to become a more skilled practitioner, a researcher, or a college professor. Other graduates move on to professional studies in medicine, dentistry, law, and management. Others may directly enter areas such as sales or management.

Most computer science graduates go on to technical careers in computing, at least initially. They may assume various professional positions (as illustrated in Figure 11) in business, industry, academia, government, or research institutions. The computer professional may sometimes work alone, but often they will be part of a multidisciplinary team that must analyze a problem, devise approaches to a solution, select the best solution, and then design, implement, test, install, and support a computer-based product that solves the problem. Figure 12 illustrates how various job roles might cooperate in the development of a software product. Figure 13 shows the relationship between various support roles and software product development.

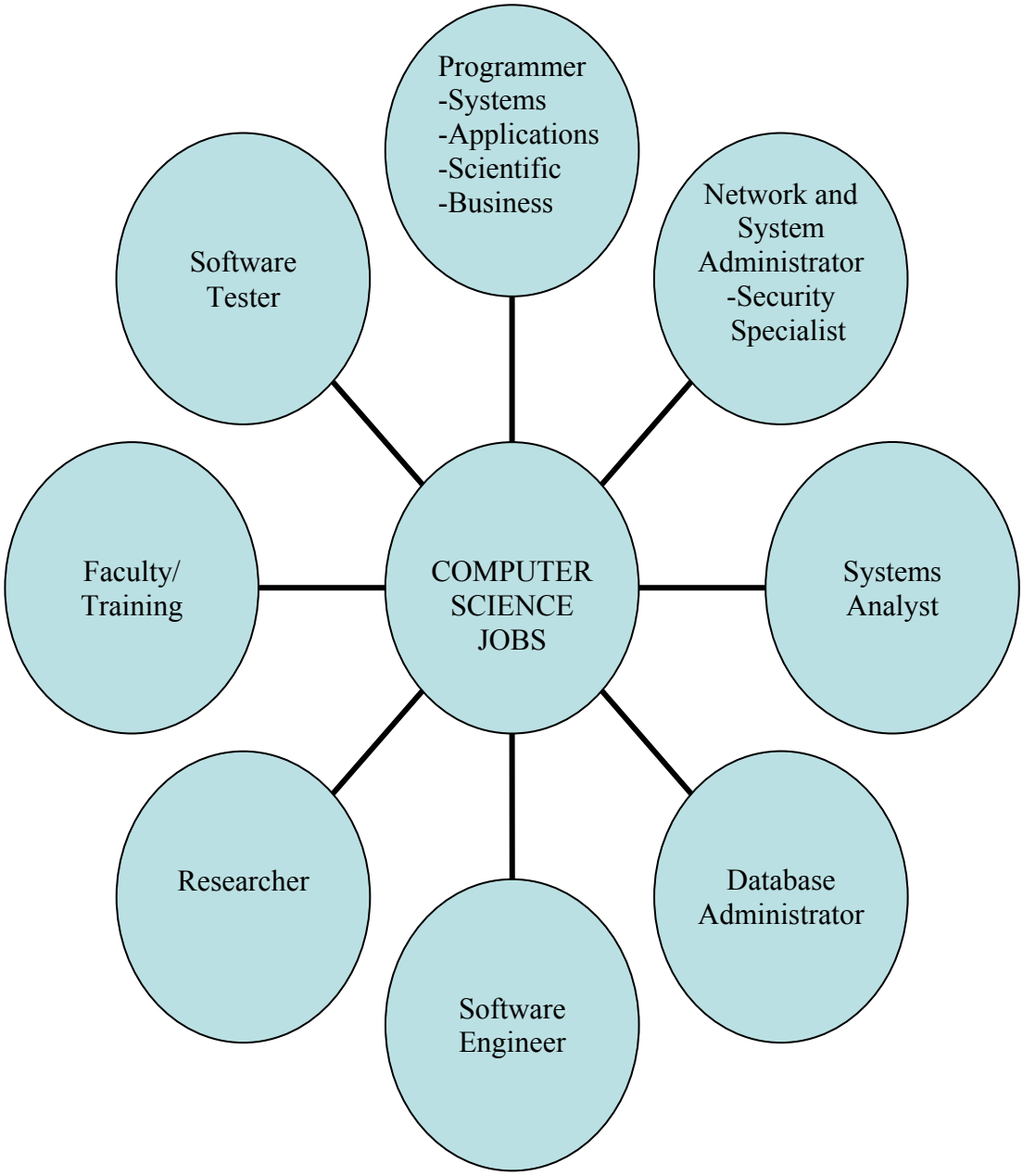

Figure 11. Selected Job Roles for Computer Science Graduates

Some computing careers require considerable software development. These are mainly programmers, system analysts, and software engineers. While *programmers* are individuals with varied educational backgrounds whose main task is programming, *system analysts* and *software engineers* are more concerned with the analysis and design of the entire system. A related job that requires considerable software knowledge is a *tester* who is part of the quality assurance team in product development. A detailed description of the various job responsibilities is given in the table in Figure 14.

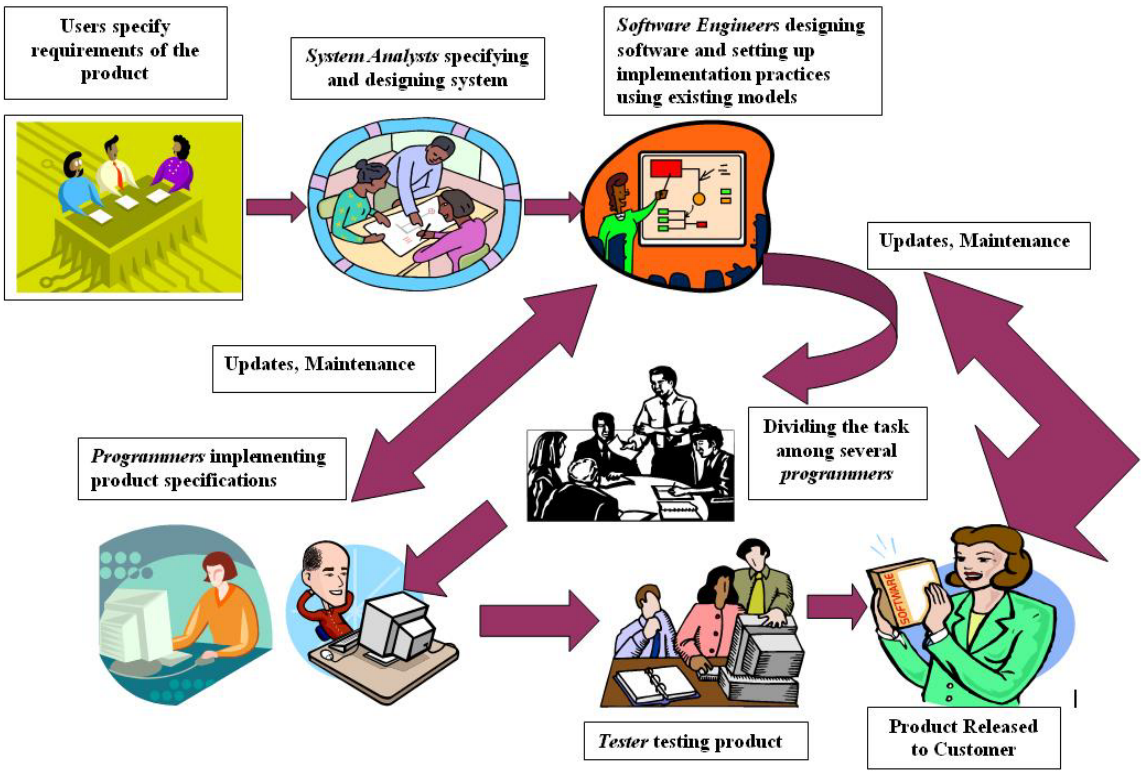

Figure 12. Computer Professionals in Software Product Development
[Microsoft Word Clip Art Components]

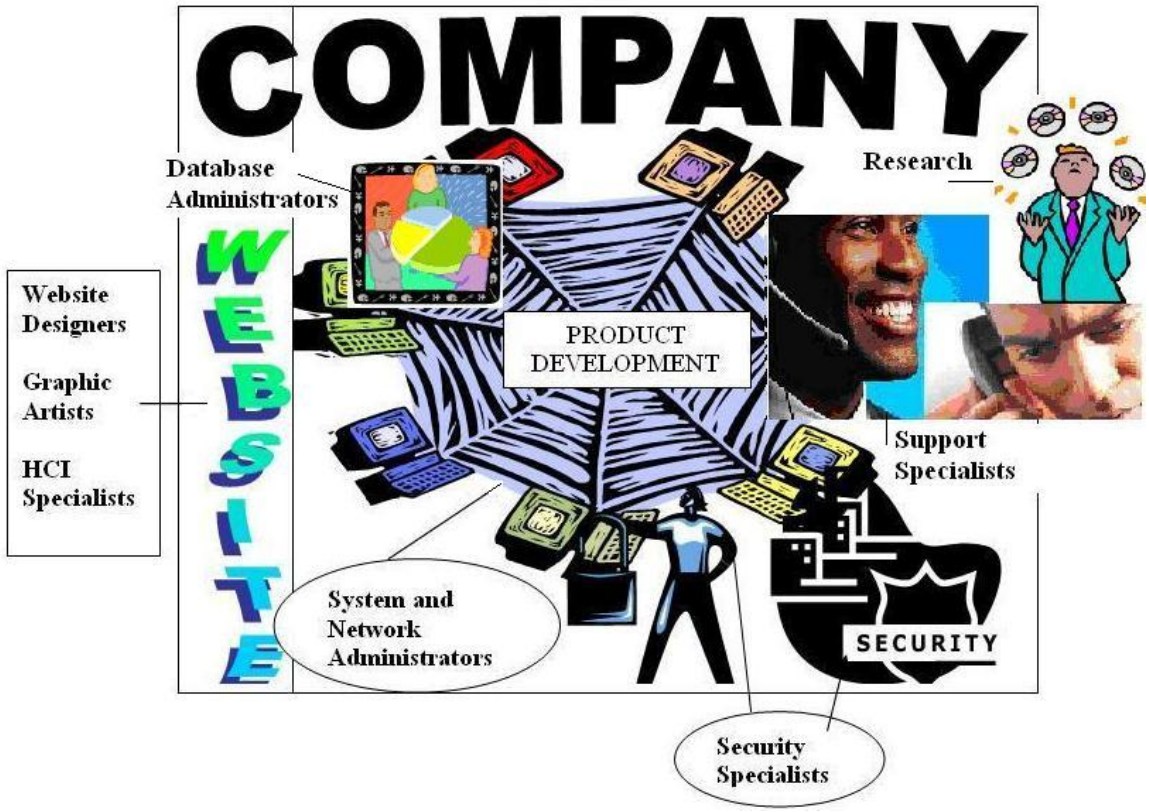

Figure 13. Computing Jobs in a Company [Microsoft Word Clip Art Components]

Job Title	Responsibilities
Systems Analyst	• Discuss system requirements with users • Define system to satisfy user needs; specify hardware and software requirements for system • Possess good communication skills • Have extensive knowledge of both technical capabilities and business organization
Software Engineer	• Design and build software using existing models • Analyze user's needs; design, implement, test, and maintain large software systems • Apply software practices to ensure product reliability, cost effectiveness, and security • Responsible for entire "life cycle" of a software system
Programmer	• Design, write, test, and maintain programs for a variety of applications
Software Tester	• Design and implement new test cases to test product functionality • Assess functionality and integrity of product by carrying out testing on different platforms • Identify errors and report required changes in software

Figure 14. Software Development Jobs

Typically system analysts and software engineers begin their careers with an initial focus on programming and as they gain more experience, they move into the more planning oriented roles. A doctorate in computer science may enable one to move directly into system design roles based on research experience. These roles require a thorough understanding of system and software design principles that can be gained through experience or research.

Other computing careers require technical hardware and software knowledge but do not require extensive software development. These include network and systems administrators, database administrators, security specialists, and support specialists as shown in Figure 15.

At this point, it is important to consider the difference between an education in computer science and engineering and technical training. The former, in the setting of a four-year college, seeks to lay the foundation of attitudes, methods, knowledge, and skills for a long, productive career as a scientist and engineer. The latter seeks to prepare the students for jobs as technologists or technicians by teaching them the practical skills and knowledge needed for an immediate job. Good technicians are needed to keep society functioning, but it is the scientists and engineers who help design the future.

Job title	Responsibilities
Network and System Administrators	• Design, install, and maintain organization's computer network • Maintain network hardware and software • Implement and maintain network security measures • Install computer systems software • Manage computer systems • Plan for future network growth
Security Specialists	• Install and maintain secure systems • Specify security rules and procedures • Develop methods and install software to prevent unauthorized access • Monitor log files for suspicious activities
Database Administrators	• Maintain large data bases • Responsible for performance, access control, data integrity, and database security
Support Specialists	• Provide technical assistance, support, and advice to customers and other users • Answer telephone calls; analyze problems using automated diagnostic programs, and resolve recurrent difficulties • Present as help desk technicians or customer service representatives

Figure 15. Other Technical Computing Jobs

There has been considerable concern in recent years that all the good computing jobs are leaving the United States to go to off-shore locations. This concern is exaggerated. It is the case that jobs have been "out-sourced" to such locations. This will continue to be the case because the general computing infrastructures, knowledge, and skills have greatly improved in a number of countries in the world and will continue to do so. However, many of these "off-shored" jobs are in areas such as customer support centers and well-defined programming and testing tasks. Jobs that require physical presence and/or an intense understanding of the business and national culture cannot be readily out-sourced. These include jobs such as systems and network administration, systems analysis, and database administration. Businesses are less likely to out-source high-profile roles that are of strategic importance to the future success of the company. These highly skilled positions will likely stay close to the home office of the company. These are jobs such as security specialists, system architects, and the software engineers who define and manage the work undertaken on an out-sourced basis. Also a number of new areas are developing at the interface between computing and other disciplines in which the developed world will continue to have an advantage for some time—areas such as *bioinformatics* and *multimedia* technologies. American programmers can no longer rely on simple knowledge of a programming language and a few other technologies to assure them of high paying jobs. However, strong high-level systems analysis and software engineering skills will continue to be in demand in the United States.