October 1989

Adapting to Computer Science

Saul Gorn
University of Pennsylvania

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-89-61.

Adapting to Computer Science

Abstract

Although I am not an engineer who adapted himself to computer science but a mathematician who did so, I am familiar enough with the development, concepts, and activities of this new discipline to venture an opinion of what must be adapted to in it.

"Computer and Information Science" is known as "Informatics" on the European continent. It was born as a distinct discipline barely a generation ago. As a fresh young discipline, it is an effervescent mixture of formal theory, empirical applications, and pragmatic design. Mathematics was just such an effervescent mixture in western culture from the renaissance to the middle of the twentieth century. It was then that the dynamic effect of high speed, electronic, general purpose computers accelerated the generalization of the meaning of the word "computation" This caused the early computer science to recruit not only mathematicians but also philosophers (especially logicians), linguists, psychologists, even economists, as well as physicists, and a variety of engineers.

Thus we are, perforce, discussing the changes and adaptations of individuals to disciplines, and especially of people in one discipline to another. As we all know, the very word "discipline" indicates that there is an initial special effort by an individual to force himself or herself to change. The change involves adaptation of one's perceptions to a special way of viewing certain aspects of the - world, and also one's behavior in order to produce special results. For example we are familiar with the enormous prosthetic devices that physicists have added to their natural sensors and perceptors in order to perceive minute particles and to smash atoms in order to do so (at, we might add, enormous expense, and enormous stretching of computational activity). We are also familiar with the enormously intricate prosthetic devices mathematicians added to their computational effectors, the general symbol manipulators, called computers.

Comments

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-89-61.

Adapting To Computer Science

MS-CIS-89-61

Saul Gorn

Department of Computer and Information Science
School of Engineering and Applied Science
University of Pennsylvania
Philadelphia, PA 19104-6389

October 1989

Adapting to Computer Science

Saul Gorn
Computer and Information Science Department
School of Engineering and Applied Science
University of Pennsylvania
Philadelphia, PA 19104

1989

Remarks on the central concepts, paradigms, and areas of Computer Science, the identification of those requiring an engineering approach, and the specification of which of these are novel to engineers.

Presented to the "Workshop on Adaptability Among Engineers", at The National Academy of Sciences. September 29, 1989

Contents

1 Introduction

Although I am not an engineer who adapted himself to computer science but a mathematician who did so, I am familiar enough with the development, concepts, and activities of this new discipline to venture an opinion of what must be adapted to in it.

"Computer and Information Science" is known as "Informatics" on the European continent. It was born as a distinct discipline barely a generation ago. As a fresh young discipline, it is an effervescent mixture of formal theory, empirical applications, and pragmatic design. Mathematics was just such an effervescent mixture in western culture from the renaissance to the middle of the twentieth century. It was then that the dynamic effect of high speed, electronic, general purpose computers accelerated the generalization of the meaning of the word "computation" This caused the early computer science to recruit not only mathematicians but also philosophers (especially logicians), linguists, psychologists, even economists, as well as physicists, and a variety of engineers.

Thus we are, perforce, discussing the changes and adaptations of individuals to disciplines, and especially of people in one discipline to another. As we all know, the very word "discipline" indicates that there is an initial special effort by an individual to force himself or herself to change. The change involves adaptation of one's perceptions to a special way of viewing certain aspects of the world, and also one's behavior in order to produce special results. For example we are familiar with the enormous prosthetic devices that physicists have added to their natural sensors and perceptors in order to perceive minute particles and to smash atoms in order to do so (at, we might add, enormous expense, and enormous stretching of computational activity). We are also familiar with the enormously intricate prosthetic devices mathematicians added to their computational effectors, the general symbol manipulators, called computers.

The disciplines require us, perhaps less dramatically but always more subtly, to change the way we understand, how we act, how we act in order to understand, what we should understand in order to act, etc. The adaptation of a discipline over a period of time is described by its history; this is sometimes slow, in the human scale, over the millenia, and sometimes so fast for humans as to be considered revolutionary, as in our two examples, or, more crucially, in the scientific revolutions whose structure was studied by Thomas S. Kuhn.

Because of this concern with adaptability of individuals and disciplines, it is useful to consider a rough taxonomic subdivision of disciplinary attitudes. Just as the individual's perception and behavior is a result of flow of signals, one way from receptors to perceive, and other of commands to effectors to act, so are there two types of disciplinary signals; there are descriptions of aspects of the world, presented in declarative sentences, and there are prescriptions of what ought to be done about those aspects, in imperative sentences. Theories are logically or empirically organized systems of the former, and programs are organized systems of the latter. Some disciplines are mainly concerned with producing theories in what we might call their "object language", reserving their programmatic statements to the discussion of their methodology, in what we might call their "meta-language"; examples are pure mathematics (where the methodological language is that of logical programs), and the empirical sciences (where the methodological language is that of experimental programs). Let us call such disciplines "knowledge-oriented". Other disciplines are mainly "action-oriented" and concerned with programs, such as hunting, cooking, manufacturing, constructing, etc. Still other disciplines have as their purpose applying knowledge to achieve action. The professions of law, medicine, and engineering are examples. These last must be sensitive to both the changes in knowledge and the changes in activities; they therefore change more than the knowledge-oriented and the action-oriented disciplines they are bridging, and if one of these goes through a revolution the connecting professional discipline may well follow suit. In any event, let us call changes in a profession caused by changes in the disciplines bridged the "professional change syndrome".

A young discipline, like a young person is an effervescent mixture of all three of the above

mentioned types of activity. Until the mid-twentieth century, mathematics and physics were like this. Computer Science is like this now.

As time goes on, the action of perceiving for the discipline may disturb what is being perceived, especially if the perceiving and perceived elements are the same order of the magnitude; this was the case with particle physics and quantum mechanics, resulting in the indeterminacy principle. More generally, the knowledge-oriented and action-oriented aspects of the young discipline will conflict, putting the discipline through a crisis, and possibly a revolution. This has happened to mathematics. Engineering must adapt to both the change in mathematics and the change in physics, two examples of the professional syndrome in engineering.

Since the success of engineering, or engineering aspects of mathematics, physics, and computer science, is measured by how many problems it solves, there is a further cause of change, beside the professional syndrome, one that we might call the "solved problem syndrome". Here the result may be a standard technique that no longer belongs to the profession but is either relegated to the discipline that presented the action problem - for example all of applied mathematics - or the result may become a new separate discipline. In the case of engineering an old example of the latter is plumbing. In transportation such results have been the train - "engineers", flight - "engineer", etc. Somewhat similar is the engineering application of x-rays and electro-magnetics to medicine, resulting in Radiology. The latter is still changing, having developed CAT-scans, PET-scans, magnetic resonance techniques, and, by another application of computer techniques, three-dimensional graphic imaging.

To see how engineers must adapt to computer science, we will first see how the engineering aspect of mathematics was originally computational. Then we will see how those computational aspects of mathematics developed to involve not only physical engineering but also psycho-linguistic aspects. This will lead us to enumerate the areas of computer science, each of which alone has a formal, an empirical, and a design aspect. Some of the formal aspects will be novel to engineers, adding to the mathematical background a computer engineer requires beyond the analytic background all engineers need; but all the design aspects will of course be engineering oriented. What is more, so many of these design aspects will be, not physical, but rather psycho-linguistic and hence, close enough to symbolic and human activities to be most novel to engineers. These will require the greatest adaptation.

2 How Mathematics Separated into Pure, Applied, and Computational

Because mathematics arose from the shepherd's need to count, the agricultural need for measurement and astronomical information, and the urban commercial need for both, calculation and geometric construction techniques were, of course, ancient. That summation of Pythagorean mathematics, the "Elements" of Euclid, therefore contained those primitive programs, the constructions with straight edge and compass, and those sophisticated verifications of the correctness of those programs, the formal geometric proofs. Euclid's students could therefore inscribe in a circle regular polygons with 3, 4, 5, 6, 8, 10, 12, 15, 16, 20, etc. sides, thereby computing lengths – considerably more than agriculture required. Meanwhile the Greek, and later the Roman, worlds made mechanical use of counting boards (abaci) and counters (calculi) to do their counting and sums.

However, it was not until the eighth century A.D. that the western world could begin to profit from the Hindu programming of numerical computation. It was then that Al Khowarizmi in the Arabic world made specific the algorithms of algebra; and it was only several hundred years later that the Arabs picked up the Hindu numeral notation itself. It was therefore only in the renaissance that the numerical symbolism and its resulting algorithms came to be fully developed. But it already

had gone beyond simple numerical calculation into algebraic and trigonometric calculations – for astronomy as well as for commerce. Even so, the arabic-hindu numerical data structures and the corresponding algorithmic programs for addition and multiplication did not come into **common** use until the eighteenth century!

Thus, from the renaissance until the mid-twentieth century, at a leisurely but slowly accelerating pace, mathematicians kept expanding their symbolic notations via their algebraic expressions, number theoretic expressions and functional expressions, and computed expressions that were formal derivatives or exact solutions in terms of the elementary functions to differential and other functional equations. They were expanding their language of descriptive, implicit and prescriptive, explicitly programmed specifications. Solving equational problems meant converting from the first type, declarative to the second, imperative specification languages. Note that this was the reverse of the Euclidean verification of the imperative construction procedure by the declarative language of formal proofs that we mentioned before.

At the same time that mathematicians were doing notational and computational innovation, they were simultaneously developing informal theory, and applying both to physics and astronomy (these were the three types of activity we mentioned in our introduction). To name only a few in this development over the centuries, there were Vieta, Fermat, Descartes, Pascal, Newton, Leibnitz, Euler, the Bernouillis, Lagrange, Laplace, Cauchy,... let us stop at Gauss just to see how much the different activities spread within a single individual. Gauss was interested not only in pure mathematics – the fundamental theorem of algebra, Number-Theory, and differential geometry -, but also in applied mathematics – Astronomy and Physics –, and in symbolic invention and new extensions to computation – the congruence notation in number theory, algorithms for solving systems of linear equations, the method of least squares, and error statistics. In these last activities he was a formidable calculator and programmer. In fact it was his early developed theory of ruler and compass construction, and his specification of exactly which regular polygons could be so constructed, and his program for doing it specifically for the regular polygon with 17 sides (when he was 19 years old) that made him decide to become a mathematician rather than a linguist. He had also been interested in philology.

Some of Gauss's predecessors had also been famous as philosophers, Descartes, Pascal, and Leibnitz for example. These also concerned themselves with logic and the theory and practice of computation.

For example Leibnitz, on the one hand, invented the differential quotient and integral notations and informally derived the computational rules of the differential and integral calculus, and, on the other he described a formal algebra of logical 'and', 'or' and 'not' as well as predicted the future formal logical algebra that he called a "universal characteristic". We will say more about the future of this prediction shortly.

But, in any event, by the end of the eighteenth century the informal concepts of 'infinitesimal', 'infinite summation', 'continuity', and 'differentiability' became disturbingly paradoxical; and yet they had such fertile applications to physics that something had to be done to verify what could be verified in the symbolic procedures, and to formalize them so as to safeguard them from paradoxical misuse. The whole nineteenth century was occupied with this goal.

This kind of problem was not new. The Pythagoreans had such great success in their reduction of measurements to counting, by the use of fractions, that they were confounded by the seeming paradox that the diagonal of a square could not be "rationally" related to the use of its side as a unit. The solution of this early mathematical crisis was the geometric theory of ratio and proportion and its application to the theory of similar triangles as given in the tenth book of Euclid; neater construction theories of the "real numbers" had to wait until the theories of Cantor, Dedekind, and Weierstrass in the later half of the nineteenth century.

The Pythagorean failure of the possibility of measuring by simple counting was only partly

solved by the invention of fractions and real numbers. There still remained logical confusion in language; for example Zeno's paradox of Achilles and the Tortoise not only concerned itself with an infinite series having a finite limit, but also muddied the water by setting up a confusing correlation of a particular linguistic description of the race between the two with the actual occurrence–the time taken to describe the race exactly had nothing to do with the time taken to run it. The nineteenth century development of a formal logic even more severe than those employed by Aristotle and Euclid brought out more mathematical crises and logical paradoxes.

For example, Boole in the early nineteenth century developed a symbolic calculus that he called "laws of thought". One such law was the one that medieval students of Aristotle called "modus ponens". It states that if we have two declarative sentences, call them p and q, and if we know that the sentence p is true, and also that the sentence of the form 'if p is true, then so is q' is also true, then we can conclude that the sentence q is also true. Boole had a neat algebraic symbolization of this procedure, an algebraic expression: $(p\&(p\rightarrow q)\rightarrow q)$; this is, however, a functional expression in a Leibnitzian calculus of two-valued logic; it has the value T (for "true") for each of the four possible combinations of values T and F (for "false") that p and q can have. Such a function is called a tautology (i.e. always true). Lewis Carroll at the end of the century in a paper called "What the Tortoise said to Achilles", pointed out that to achieve the result of modus ponens would require not merely this tautology but also one that states that p and $p\rightarrow q$, and it, $(p\&(p\rightarrow q)\rightarrow q) = A_1$, imply q, i.e.

$A_2 = (p\&(p\rightarrow q)\&A_1\rightarrow q)$, and further that
$A_3 = (p\&(p\rightarrow q)\&A_1\&A_2\rightarrow q)$,and, etc.

In other words, it takes an infinite number of these tautologies to achieve modus ponens.

The law of thought, modus ponens, is **not** in the Boolean expression language; rather it is a **valid procedure** in a **prescriptive meta-language** that talks about the production of proofs in the **Boolean object language**. The tautologous symbolic expression had been designed to symbolize modus ponens, and the symbol was confused with what it symbolized. The tautology is no more a law of thought than the artist Magritte's picture of a pipe, that he entitled "this is not a pipe", is a pipe!

In general, crises in mathematics were signaled by paradoxes. The paradoxes were usually resolved by finally recognizing that they were either 1. - Reductio ad absurdum proofs that something did not exist,
or 2. - Reductio ad absurdum proofs that some goal was impossible to achieve by a finite means, or
that 3. - mixing meta-language with object language caused us to confuse symbols with what was being symbolized.
And even in the reductio ad absurdum solutions, the main problem often was to find out what was presumed to exist but did not, or why it did not. For example 'the real number $\sqrt{2}$ is **not** rational' or 'the number of prime numbers is **not** finite'.

Many of the dramatic advances were due to lifting the level of thinking by meta-linguistic description, and by generalization at the meta-language level. For example-algebraic symbolization is a generalization into meta-language of the object language of arithmetic – and analytic symbolization is a generalization into meta-language of the algebraic language of arithmetic functions (not all functions, are algebraic, or even representable by infinite series of algebraic expressions). But the symbolic procedures suggested at meta-language symbol level might not have valid meaning in the operations at object language level. The mixture of object language and meta-language is intuitively powerful, but, like most powerful tools, is dangerously capable of misuse. This was why mathematics went even further than Euclid in carefully formalizing proofs. Archimedes and Euler intuitively handled infinite series correctly, but too many paradoxical operations, such as hidden

divisions by zero in algebraic meta-language or its equivalent in continuity and differentability discussions made careful logic necessary.

And then the paradoxes even turned up in the logic!

Cantor, for example, gave a formal definition for two sets having the "same number of elements"; it was later used to give a formal definition of cardinal and ordinal numbers, even for infinite sets. He was able to show that there were just as many rational numbers between zero and one as there were natural numbers all told, i.e. that the rational numbers are **countably infinite**; a seeming paradox, but not really one; and he capped this result with a reductio ad absurdum proof by his 'diagonal method' to show that the number of real numbers between zero and one is **not** countably infinite, i.e. has a **larger** infinity than the number of natural numbers.

All this activity of formalization to avoid paradoxes and contradictions was resulting in an extension of the Pythagorean program of reducing all mathematics to numbers; it was all being reduced to a formal theory of sets, and then by Frege, and later by Whitehead and Russell, to formal logic itself, as foreseen by Leibnitz. But, the logic itself emerged with dangerous paradoxes, beyond what Lewis Carroll had recently found in Boole's laws of thought. All this thinking about thinking, and mixing meta-language with object languages, seemed fraught with dangers due to self-referencing: the numbers of sets of numbers, sets of sets, etc.

In fact Russell confounded Frege, immediately after Frege's publication with his famous paradox of "The set of all sets that are not members of themselves"(is it or is it not a member of itself ?!?!). And some further paradoxes brought to the surface the relationships to language. For example there was the Richard paradox, which first pointed out that there could be only a countably infinite number of English phrases that specify functions of one natural number variable, and then used Cantor's diagonal argument to show that this could not be true. And then there was Berry's paradox: "The least natural number not nameable in fewer than twenty-two syllables" is hereby named in twenty one syllables.

Thus, by the beginning of the twentieth century, a larger and larger portion of mathematical activity was the axiomatization and presentation of mathematical theory in an even more rigid formalization than Euclid's elements. This area was called 'pure mathematics'. And Hilbert's program for the development of pure mathematics was to axiomatize and present a formal system that would cover all possible mathematical truths; any properly (i.e. mathematically) stated sentence in such a system would be formally provable or else formally contradictable. This optimistic program was formally proved to be unattainable toward the middle of the century; Godel showed that any sufficiently rich formal system was either inconsistent or incomplete in the sense that there would be statements within it that were undecidable, i.e. could neither be proved nor disproved. His proof was something like Richard's paradox, using a Cantorian diagonal argument on a formalization of a meta-language. At about the same time logicians were uncovering a number of undecidable questions and unsolvable problems by developing a number of equivalent theories of computation that we will consider below.

All through the history of mathematics the word "computation" had been expanding its meaning. We will shortly examine the process in more detail. The formalization process of the nineteenth century accelerated it by including all the algebras, geometries, and logics. But it was the necessity to formalize the programming of computations caused by the inhuman speeds of the recent general purpose computers that made computation include all precisely specifiable symbol manipulation. Now general symbol manipulation includes the pragmatic effects of deliberate ambiguity caused by shifting interpretation, for example between object language and meta-language. General computer programming must therefore include what mathematical logic programming must forbid. Again, the mathematical study of arithmetic is independent of the way numbers are represented; the computer program for addition depends not only on the particular computer type, but also on the number representation system used. There are so many pragmatic questions in computer theory,

use, and design that the area of computation and the area of mathematics were pulled apart.

For a different reason, mainly what we called the solved problem syndrome, applied mathematics also separated from pure mathematics.

By the middle of the twentieth century the applications of mathematics had spread to so many different disciplines that the spread was too vast for any single person's curriculum. The mathematical applications, once the mathematical aspect of their disciplines had been precisely specified and handled, were relegated to the discipline of the application. A few universities were concerned with a united applied mathematics group, but, by and large, the discipline became restricted to "pure mathematics".

3 How Computation broadened its Scope and developed a close relationship with Engineering, Logical Theory, and Psycho-Linguistics

Mechanical aids for counting became a necessity especially in urban cultures, for example around the Mediterranean in Hellenic and Roman times. And from those beginnings computation, like mathematics generally, developed in three interacting strands:

First there was the development of the physical aids and their psycho-linguistic aspects into machines; the physical aids development is the obvious engineering aspect, the psycho-linguistic the more subtle one, as we shall see.

Then there was the theoretical aspect-beginning with mathematics as a whole, but narrowing down to formal proofs, and then to formal logic itself.

And thirdly, forming a spiral, there was the psycho-linguistic development of naming and specifying the **data structures** (at first, the number representation systems) and **the expressions of the procedures** for handling them (their programs or algorithms).

During the interacting development of these three phases in computation, it was the expressions of the procedures that were turned into machines. For example, as mentioned before, in Hellenistic and Roman times, counting and adding were facilitated by counting boards with lines on them for placing counters. These static devices, made mobile by fleetly calculating human fingers, were of two designs; the first, a simpler design but more bulky, had room for ten counters per line, while the second, a more compact one, had five counters per line, and intermediate lines designed for at most two counters to indicate the number of full hands of five fingers that had accumulated on the base lines. The description of the counting and adding procedures for the bulky counting board was, of course, much simpler.

The two types of roman and chinese abacus are further developments of these placid machines, where the lines are replaced first by slots and then by wires, and the counters are beaded in or on them vertically instead of being strung out horizontally; in the more compact type the intermediate lines are vertically above the ones they relate to.

The Roman symbolization of the total number counted on a board was a compact description of the picture on a compact board, using I and V for the first line and its associate, X and L for the second, C and D for the third, etc., and a reversing grammar to obtain extra compactness, IV for IIII, XL for XXXX, etc. The parsing of these number representation phrases was fairly complicated in order to achieve compactness and representation on one line.

The Hindu-Arabic symbolization, especially when a symbol for an empty line, zero, was introduced, was a compact description of the bulkier abacus. This second **phrase structure language** used the position of the numeral to represent the position of the wire and thereby avoided the necessity of using a larger and larger alphabet for more and more wires. Moreover, the algorithm for addition in this language had a considerably simpler description than did that for addition in

the roman numeral phrase structure language.[1]

It would seem that it took mankind a thousand years to decide that the descriptive language for the bulkier machine was more useful. The algorithm for addition in the arabic symbol system has a much simpler description as well as a need for only a finite alphabet.

The young Pascal (19 years old) assisted his tax-collecting father by designing a more mobile mechanical adding machine than the abacus. It simulated in hardware the program for addition in the Arabic system. And within a generation Leibnitz adapted the same technique to make the machine do multiplication, albeit with the vigorous help of the human hand, i.e. some software was still necessary.

The explosion of functional equation solving techniques in mathematics during the next two hundred years made a more Gauss-like calculator desirable. At the beginning of the nineteenth century Charles Babbage felt that all function table construction should be made mechanically and automatically, and he spent some decades designing a "difference engine" that would do the job. He was also convinced that a device similar to the cards controlling a Jacquard loom in textile weaving could be used to allow the descriptions of many procedures to serve as different programs in one and the same "analytic engine". Although his mechanization ambitions were defeated because technology was not advanced enough to standardize the necessary parts, these ambitions were prophetic enough to anticipate Hollerith's punched card machines at the end of the century, and the electro-mechanical relay machines of Konrad Zuse, Howard Aiken, and George Stibitz a hundred years after his time. In effect Babbage had invented the concept of programming general purpose calculating machines and the idea of programming languages for the specification of how to run numerical algorithms. His message was kept alive not only by the designers of machines in the forties and fifties of this century in the United States and Britain, and onto the European continent, including the Russian mathematicians and engineers following Tchebycheff, but also by the logicians in the thirties, forties and fifties. These latter developed a general theory of computation even before electronics took over.

We therefore return to the portion of the history of mathematics where an extension of the concept of computation was occurring in the nineteenth century. We have already noted that following al Khowarizmi a meta-language for arithmetic, namely algebra, appeared and flourished. The solution of algebraic equations as explicit functions of their coefficients was sought. Where the solutions were impossible in the number systems already available, new extensions to the number concept were invented. We have already remarked on this happening when the Pythagoreans had to deal with that anachronism of the future, $x^2 = 2$. Then, through the renaissance, it resulted in the invention of the next imaginary class, the negative numbers, yielding the solution to all equations of the form $ax + b = 0$, except, of course, for $a = 0$. Then a unified solution to all quadratic equations, $ax^2 + bx + c = 0$, was found as a function of a, b, and c, provided the invention of numbers was extended to include another imaginary class, first called imaginary numbers, and finally accepted as the "complex numbers"; and the unified solution called for addition, subtraction, multiplication, division, and, the Pythagorean crisis having long been understood (?), the operation of taking a square root. The program, or algorithm for the construction of the solutions were presented explicitly in an algebraic form. What could be said about all algebraic equations? Gauss finally proved that the complex numbers would suffice to solve them, but not by an algebraic formula. (We would now say that "the complex number system is algebraically closed".) By his time the

[1] Because the natural numbers form a simple chain, the same linguistic expressions, the number namers, serve the double purpose of being enumerators – while the counting proceeds – and summarizers – when the counting ends. When we want to stress this difference we use ordinal language instead of cardinals; their isomorphism is a deep property of finite natural enumeration. This isomorphism is lost in both infinite cardinals versus ordinals, and in 'tree names' as against 'tree addresses' in ramified structures such as decision trees, family trees, classification systems, or organization charts.

construction of the solution by addition, subtraction, multiplication, division, and the extraction of roots had been achieved for degrees 3 and 4, and no more.

It was then that Galois, by studying the effect of permuting the roots of an equation, discussed the algebraic systems called groups. He found that those and only those equations were "solvable by radicals" whose Galois group has a certain restricted structure; and he further found that the general algebraic equation of degree greater than or equal to five did not have this structure, called, incidentally, "solvability". Thus another condition for solvability and unsolvability emerged, as had those that prompted the invention of extensions to the number system, or Gauss's condition for solvability by straight edge and compass construction.

Computation now included combinatoric ones concerned with rearrangements of objects, and counting arrangements fulfilling various conditions. The theory of probability had already depended upon such combinatoric processes in the study of games of chance, at least since the time of Fermat and Pascal; And Euler had considered counting and classifying paths on road maps or on edges of polyhedra in the beginnings of combinatory topology known as Analysis Situs"; and the specification of the new algebraic groups themselves appeared in a study of strings of characters, each character representing a generator, where certain pairs of strings were identified as producing the same group elements. Computation now included the purely syntactic study of certain types of symbol manipulation (ignoring the symbolism); its application to group theory and a variety of algebras, geometries, and topologies were studied through the nineteenth century by, for example, Hamilton, Cayley, Poincare, and Burnside. At the outbreak of World War 1, Thue was publishing papers on "word problems" that were fore-runners of some of the general theories of computation. Cantor's diagonalization proof was a syntactic game with number representation words. The logical paradoxes were semantic games with words as well. Logicians of the Polish school, their student Herbrand, and Gentzen were also thinking in terms of syntactics and semantics of such symbol manipulation games. Finally, after Godel's work that we mentioned above, the logicians of the thirties and forties followed three types of approach (and their mixtures) to a general theory of computation. They all produced important examples of unsolvable problems, showing the limits of such general computation; and the theories were proved to be equivalent.

The first approach presented purely syntactic "rewriting systems" as in the "presentations" of groups, and the generalization toward word problems by Thue. Post's "production systems" or "semi-Thue" systems was an example, and Post presented an undecidable problem, the "correspondence problem". Some years later it was shown that the word problem for semi-groups and groups, namely determining whether two products of generators represented the same element, was undecidable. In this class, some years later, was Markov's theory of algorithms.

The second approach presented formalizations of the meta-language of such syntactical symbol manipulation; the characters in the meta-language had meanings (semantic content) and were therefore really symbols, where the object language characters might not be. For example, Church, in his λ-calculus, eliminated the confusion in mathematical notation between symbolization of a function and symbolization of the value of a function for a general element in its domain. Rosser concerned himself with the possibility of reducing a λ-expression to a "normal form". Curry considered primitive symbol manipulating operators, the "combinators", and their combinations. And Kleene formalized recursion to produce a theory of recursive functions. This approach has had a direct effect in recent studies of programming language semantics and in the theory of program verification.

The third approach was more pragmatic in that it specified the users or interpreters of the symbols being manipulated as well as the syntactics and semantics of the manipulations. These users and interpreters were the symbol manipulating machines, and the approach was directly descended from Babbage. The prime example of this approach was that of Turing. He specified the concept, that of Turing machine, as any mechanism that had a finite number of states and

a potentially infinite tape that, at each cell, could be read, or erased, or printed on, or switched to another cell depending on the state the machine was in, and had a switch to another state as part of the interpretation of the symbol. Prime illustrations of Turing machines were concatenators of strings of symbols, copiers, combiners, etc. A carefully designed combination of a few of these Turing showed to be universal in the sense that it could copy the specification of any special machine and imitate it. The tapes specifying turing machines served as programs for the universal turing machine. The concepts of program, procedure, algorithm, and machine had become identified as far as their effect was concerned, even though their uses of space and time might be quite different. Turing showed that it was impossible to construct a program that would accept any program and appropriate data and determine in a finite time whether or not the problem represented by the program and data would come to a halt or would continue indefinitely. During the second world war, Turing was also involved with the design and use of a machine in Britain, the Colossus, whose main purpose seems to have been cryptanalytic.

This brings us to the time of general purpose machines and electronic developments.

The electro-mechanical calculating machines were already so much faster than human computation that they required prior program preparation. Rutishauser even contemplated making the programming process itself partly automatic; he therefore foresaw the extension of the meaning of the word "computation" that the succession of generations of machines were about to add to mathematical general symbol manipulation, namely the construction of computer programs themselves.

The first electronic general purpose calculating machine, the ENIAC, was at first programmed for special problems by hand-plugging the interconnection of the registers from which and into which data was to be moved and operated on. Later a code system for setting up such appropriate hard-wire switching was entered on the tables used for storing numerical data by switches, i.e. the programs were coded numerically.

The EDVAC, immediately after the ENIAC, was designed with a fast access internal storage (mercury delay lines) of one thousand and twenty four words that could be either data or coded instructions. Effectively, the program was the switching design of the special purpose machine one wanted the general purpose machine to become; contents of a storage cell were data if they were sent to an adder, say, but were an instruction if they were sent to the "instruction register" to be interpreted (i.e. decoded, or parsed) as an instruction; and not infrequently, in a programmed loop, the first was done to change the address of the addend to be used, and immediately afterwards the second was done to the result. The machine was modifying its own instructions, as though self-consciously. This brings to mind the mixing of meta-language (i.e. the programmed instructions) and object language (the computational data).

On the one hand this common storage of instructions and data led to the construction of programmed compilers and interpreters of higher level languages; on the other hand it illustrates the principle of the "logical equivalence of hardware and software". It was later modified by the introduction of standard binary codes for the complete typewriter keyboard. The data was no longer viewed as just numerical. The similarity of these electronic symbol manipulators to the universal turing machine was now more obvious than ever.

The programming of interpreters of higher level languages now forced the self-conscious examination and imitation of psycho-linguistic processes. Meanwhile mathematics was being applied to psychology and linguistics.

The mathematical psychology, at first, concerned itself with stochastic learning models of the reinforcement type; its only effect on computation was to sharpen the programming technique of delayed random selection. In fact it was the attempt to program problem solving with its unpredictable storage requirements, and to do it on a machine with magnetic drum storage that prompted the invention of list structure and "push-down" storage techniques in this country. This led to the dynamic storage allocation we will say more about shortly, and advanced the development

of the area called artificial intelligence.

The mathematical linguistics introduced the Chomsky hierarchy of phrase structure languages. These were approximations of the linguistic procedures employed by humans in their use of natural languages, but immediately enriched the theory of computer languages. They gave us the various types of automata, finite state, push down, etc. that were needed to parse them, and models of formal languages, e.g. finite state, context-free, etc., that were useful in studying and designing computer languages and analyzing the general computation process.

At first the coding of this computation process was done by specialists – the machine coders; in addition there were the specialists (sometimes these same machine coders) who entered the problems in the machine, ran them, arranged for their reading and printing, etc. The machine time was too expensive to be slowed down by the single user; the set of problems of a number of users was batched and scheduled by the operators.

The advancement in machine technology came at a rapid rate, though, luckily, at a slow enough rate so that internal, fast access storage was still expensive and therefore too small to afford separating instructions from data. Thus, when the magnetic auxiliary storage devices, tapes and drums, were enhanced by the faster access, internal storage (the core memories), the internal storage could increase moderately; the increase was enough to permit the specialized coder to be replaced by a program, the aforementioned compiler or interpreter of some standard higher level language; it also permitted a partial replacement of the machine operator by an automatic scheduler that shared the time allotted to a number of users, their inputs, their required auxiliary storage, and their outputs. This latter software was an "operating system" that also handled the allocation of internal storage dynamically, not only to schedule the time sharing of the users but also to allocate unpredictable storage needs of the single user. There were at least two types of dynamic storage allocation that machine users needed. The first was the recursively defined functions in such higher level programming languages as ALGOL and LISP, and the second was for the unpredictable storage necessary in the automatic simulation of human problem solving in the new area of artificial intelligence that we mentioned above.

Thus the human calculating processes, even at the programming and operating level, were self-consciously examined and programmed. The programming and operating level itself was recognized as a type of calculation much like the algebraic, analytic, and symbol manipulative level.

Meanwhile the extension of programming capability to the machine user caused the explosion of computer applications to all disciplines that were sophisticated enough to develop advanced symbolic manipulation. This includes even such action oriented disciplines as sports, with their diagrammatic simulation representable in computer graphics. The market expanded rapidly and supported the hardware, software, and application research. The machines advanced in generations of from five to ten years each – the transistorized generation, semi-conductor devices, large-scale integration, and the VLSI methods now used to construct micro-processing chips the size of finger nails. Simultaneous with the drastic decrease in size came a drastic increase in speed, a drastic lowering of cost, and a drastic expansion of the market because of a drastic extension of the meaning of computation and its applications; for example, business data processing applications were once again, as in the renaissance, an important use of machines, but now with the counters, recorders, and summarizers, i.e. the large staffs of clerks, also replaced by machines.

Meanwhile, during one generation of engineers and clerical staffs, the hardware and software went through three or four generations of change. What is more, the software production is now much more expensive then the hardware, and needs fully as much engineering. Far from the secrecy about computational design and use that characterized the international and intra-national relations immediately following the second world war, it is now clear to all that it is more profitable for all to standardize types of machine design, language design, and information interchange.

The computer systems and computations, therefore, are not merely proliferating; they are

actually clustering into larger and larger networks for the interchange of such computed information. It is not merely airlines that need such "distributed system computation" but also all extended business corporations, and international academic research.

In summary, the word "computation" now means "any precisely specified process of symbol manipulation", even pictorial. When restricted to computer programming, the semantics, i.e. the meanings of symbols is restricted to the symbol manipulative. But these meanings can be expanded to visual and other prosthetic sensing and effecting devices we choose to add to the machine's data assemblers and output reactors. This is what is done in the computations for robotics. And, indeed, some of the most humanly interactive programming techniques are now object-oriented, mixing digital video graphics, digital audio sound, in distributed computation and information networks.

4 The Areas (9) of the discipline of Computation (Informatics) and the parts of each that are engineering oriented.

We have seen that until recently it was not unusual for the same thinker to concern himself with philosophy, formal theory, empirical questions and design problems. So far it would seem that informatics, unlike most other academic disciplines, requires that three types of thinking be acquired in its educational curriculum; the curricula suggested by the Association for Computing Machinery (ACM) and the Institute of Electrical and Electronic Engineers (IEEE) reflect this. The three types are formal theoretic, empirical and experimental, and design oriented types of thinking (roughly, syntactic, semantic, and pragmatic). A most recent such discussion ("Computing as a Discipline", by Peter J. Denning et. al. in CACM, January 1989) lists nine sub-areas with these three types of thinking in each. They present this "Definition Matrix for the Computing Discipline" as follows:

	Theory	Abstraction	Design
1. Algorithms and Data Structures			
2. Programming Languages			
3. Architecture			
4. Numerical and Symbolic Computation			
5. Operating Systems			
6. Software Methodology and Engineering			
7. Databases and Information Retrieval			
8. Artificial Intelligence and Robotics			
9. Human - Computer Communication			

* (We have appended their appendix to show the detailed subjects).

And they say "It is the explicit and intricate intertwining of the ancient threads of calculation and logical symbol manipulation, together with the modern threads of electronics and electronic representation of information, that gave birth to the discipline of computing". I hope the previous section has helped to illustrate this statement in greater detail.

We note that each of these nine areas has "substantial design and implementation issues". These are, naturally, in the purview of engineering. All engineering, like all professions that transform knowledge into action, was always pragmatic, both in the popular and in the technical sense (in linguistics, "pragmatics" is concerned with resolution of ambiguity by recognition of context and appropriate interpretation). Furthermore, adaptability to humans has always been an engineering

consideration, in fact the original motivation, whence some areas have made a special study of it - e.g. systems engineering, industrial engineering, ergonomics (i.e. man-machine interaction), etc.

But it was always the relationships of material objects and surroundings to humans that was involved; what about the design of procedures, of general and special purpose languages, of automatic scheduling of distributed devices (including computers) for a distributed audience, of editing systems, of types of communications, of types of graphics, of human sensory reactions to types of prosthetic sensing, and the like? Surely aeronautical and space engineering have been concerned with the last of these. But have we designed types of thinking before, and prepared the world at large to live with them?

Before going on to consider what engineers have to adapt to because of computer science (or within computer science) let us remark on how stable the contents of computer science is likely to be; for, just as mathematics has changed into (almost) a pure "knowledge-oriented" discipline, might not computer science do likewise?

Because so much of it remains "action-oriented", namely programming and the fact that it is equivalent in a way to machine design, I believe it must remain a professional type of discipline, just like engineering, which it overlaps. Moreover it will have, I believe, the "solved problem syndrome" to an even greater extent then mathematics did (with applied mathematics), because there are so many disciplines involved with symbol manipulation; there may be very many mathematical needs in other disciplines, but there are even more symbolic needs (e.g. diagrammatic plans in sports). Even in mathematics, methodological discussions concerning clumsiness or neatness of notations are examples of applied informatics! And yet the knowledge-oriented and action-oriented parts of computer and information science will still have to remain balanced because the programming activity and machine design activity must, due to their "logical" equivalence.

5 Where Engineers need Adapting in order to participate fully in Computer Science

We have remarked in the introduction that professional disciplines may be expected to change faster than those that are almost purely knowledge-oriented, or those almost purely action-oriented. we gave two types of change symptoms, the "professional change syndrome', and the "solved problem syndrome". The variety of types of engineering of the past and present is due to the variety of knowledge-oriented and action-oriented disciplines they bridge. They each changed rapidly because of the professional change syndrome and some have disappeared because of the solved-problem syndrome.

Now essentially all the engineering types have had their share of applied mathematics, both in their theory, and in their computational practice. Mostly these theories and computations have been from analysis and from statistics; and, this will naturally continue in the needs from computer science. All engineers will continue to need numerical analysis in their core education, and all will need introductory computer courses leading to computer aided design (CAD), and probably enough graphics for both displays and to replace, for example, mechanical drawing. However, in addition to these core requirements, in their early engineering education, they must still do something to keep up with the rapidly changing computer field (the professional change syndrome again). They will have to have a continuing education in general, and the computer field will have to be part of it. Due to the solved problem syndrome they must be prepared either to change over to a resulting new discipline or to switch to new problems. In either case, continuing education, whether on the job or in a sabbatical, is necessary.

For computer engineers in particular, their core requirements in mathematics must be more than the analysis courses all engineers need. They must have abstract algebra and logic as well. These

are necessary in the understanding of programming language design and programming language capabilities. They are also necessary for the understanding of the logical limits of computation, where analysis of computational complexity would still not show what is impossible. Since, like all engineers, they must concern themselves with man-machine interaction, some ergonomics should, of course, be part of their education; but, because of the psycho-linguistic effects they must be concerned with, they should go deeper into cognitive science. After all, when they design a hardware - software - peopleware system, because of the logical equivalence of hardware and software, they must decide on which parts are more efficient in hardware and which in software; but they must also be concerned, not only with machine efficiency, but with psychological interaction as well. Not only should they be concerned with reasonable interaction between the user and the rest of the system, but with the distribution of users, advisers, machines, and interactive software in extended networks.

References

[1] Bell, E.T. - Men of Mathematics - Dover 1937 (see Zeno, Eudoxus, Archimedes, Pascal, Leibniz, Euler, Gauss, Cauchy, Galois, Boole, Cantor).

[2] Luce, R.D., Bush, R.R. and Galanter, E., eds. Handbook of Mathematical Psychology - vol. II - Wiley, 1963, (Chaps. 11, 12, 13 by Chomsky, Chomsky & Miller, Miller & Chomsky).

[3] Luce, R.D., Busch, R.R. and Galanter, E., eds. Readings in Mathematical Psychology - Wiley 1963 (see Miller - The Magical Number 7).

[4] Boyer, C.B. - A History of Mathematics - Wiley 1968.

[5] Pullam, J.M. - The History of the Abacus - Hutchinson & Company - London - 1968.

[6] Kuhn, T.S. - The Structure of Scientific Revolutions - University of Chicago Press - Chicago - 2^{nd} enlarged ed., 1970.

[7] Ralston, A. - Introduction to Programming & Computer Science - McGraw Hill 1971 - (see Chapter 1).

[8] Goldstine, H.H. - The Computer from Pascal to von Neumann - Princeton University Press - 1972.

[9] Harmon, M. - Stretching Man's Mind, Mason/Charter, 1975.

[10] Randell, B. (ed.) - The Origins of Digital Computers; Selected Papers, Springer-Verlag 1975 - (see Babbage, Hollerith, Zuse, Aiken & Hopper, Stibitz, etc.).

[11] Hofstadter, D.R. - Gödel, Escher, Bach - Basic Books, 1975.

[12] Metropolis, N., Howlett, J. & Rota, Gian-Carlo; (eds.) - A History of Computation in the Twentieth Century, Academic Press, 1980, (see the Colossus,Backus, Ershov & Shura-Bura, Bigelow, Burks, Stibitz, Wilkes, Bauer, Eckert, Mauchly, Booth, Zuse).

[13] McNaughton, R. - Elementary Computability, Formal Languages, and Automata, Prentice-Hall, 1982.

[14] Shelley & Cashman - Computer Fundamentals for an Information Age-Anaheim Publishing Company - 1984.

[15] Machlup, F. & Mansfield, U. (eds.) - Forward by George A. Miller - The Study of Information: Interdisciplinary Messages; Wiley 1983 (see sections 1,2,3,4 on Cognitive Science, Computer and Information Science, Artificial Intelligence, and Linguistics: also the equivalent of section 2 in Knowledge; vol. 4, no. 2, December 1982, pp. 164 - 251).

[16] Wood, D. - Theory of Computation, Harper & Row, 1987.

[17] Denning, P.J., Comer, D.E., Gries, D., et. al. - Computing as a Discipline - and the Appendix in Communications of the ACM, vol. 32, no. 1, January 1989.

[18] Penzias, A. - Ideas and Information - Managing in a High-Tech World - W.W. Norton, 1989, esp. chaps. 2,3,4.