

CHAPTER 1: SEVEN VIEWS OF COMPUTER SYSTEMS

A computer is determined by many factors, including architecture, structural properties, the technological environment, and the human aspects of the environment in which it was designed and built. In this book various authors reflect on these factors for a wide range of DEC computers - their goals, their architectures, their various implementations and realizations, and occasionally on the people who designed them.

Computer engineering is the complete set of activities, including the use of taxonomies, theories, models, and heuristics, associated with the design and construction of computers. It is like other engineering, and the definition that Hamming once gave is especially appropriate: engineers first turn to science for answers and help, then to mathematics for models and intuition, and finally to the seat of their pants.

In the few decades since computers were first conceived and built, computer engineering has come from a set of design activities that were mostly seat-of-the-pants based, to a point where some parts are quite well understood and based on good models and rules of thumb, such as technology models, while other parts are completely understood and employ useful theories such as circuit minimization.

In this chapter seven views are presented that the authors have found useful in thinking about computers and the process which molds their form and

function. The views are intentionally independent; each is a different way of viewing a computer. A computer scientist or mathematician sees a computer as levels of interpereters. An engineer sees the computer on a structural basis, with particular emphasis on the logical design portion of the structure. The view most often taken by a buyer is a marketplace view. While these people each favor a particular view of computers, each typically understands certain aspects of the other views also. The goals of Chapter 1 are to increase this understanding of other views and to increase the number of representations used to describe the object of study and hence improve on its exposition. Thus, these views form a useful background for the subsequent chapters on past, present, and future computers.

VIEW 1: Structural Levels of a Computer System

In Computer Structures [Bell and Newell, 1971], a set of conceptual levels for describing, understanding, analyzing, designing, and using computer systems was postulated. The model has survived major changes in technology, such as the fabrication of a complete computer on a single silicon chip, and changes in architecture, such as the addition of vector and array data types.

As shown in Figure 1, there are at least five levels of system description that can be used to describe a computer. Each level is characterized by a distinct language for representing the components associated with that level, their modes of combination, and their laws of behavior. Within each level there exists a whole hierarchy of systems and subsystems, but as long as these are all described in the same language, they do not constitute separate levels. With this general view, one can work up through the levels of computer systems, starting at the bottom.

The lowest level in Figure 1 is the device level. Here the components are p-type and n-type semiconductor materials, dielectric materials, and metal formed in various fashions. The behavior of the components is described in the languages of semiconductor physics and materials science.

The next level is the circuit level. Here the components are resistors, inductors, capacitors, voltage sources, and nonlinear devices. The behavior of the system is measured in terms of voltage, current, and magnetic flux. These are continuously varying quantities associated with various components, hence there is continuous behavior through time, and equations (including

differential equations) can be written to describe the behavior of the variables. The components have a discrete number of terminals, whereby they can be connected to other components.

Above the circuit level is the logic level. While the circuit level in digital technology is very similar to the rest of electrical engineering, the logic level is the point at which digital technology becomes separate from electrical engineering. The behavior of a system is now described by discrete variables which take on only two values, called 0 and 1 (or + and -, true and false, high and low). The components perform logical functions called AND, OR, NAND, NOR, and NOT. Systems are constructed in the same way as at the circuit level, by connecting the terminals of components, which thereby identify their behavioral values. After a system has been so constructed, the laws of boolean algebra can be used to compute the behavior of the system from the behavior and properties of its components.

In addition to combinational logic circuits, whose outputs are directly related to the inputs at any instant of time, there are sequential logic circuits which have the ability to hold values over time and thus store information. The problem that the combinational-level analysis solves is the production of a set of outputs at time t as a function of a number of inputs at the same time t . The representation of a sequential switching circuit is basically the same as that of a combinational switching circuit, although one needs to add memory components, such as delay elements (which produce an output at time $t-$). Thus the equations that specify sequential logic circuit structure must be difference equations involving time, rather than the simple boolean algebra equations which describe purely combinational logic circuits.

A glance at Figure 1 reveals there is another part to the logic level. This part is called the RT (register-transfer) level. The components of the RT system level are registers and the functional transfers between those registers. The functional transfers occur as the system undergoes discrete operations, whereby the values of various registers are combined according to some rule and then are stored (transferred) into another register. The rule or law of combination may be almost anything, from the simple unmodified transfer ($A \leftarrow B$) to logical combination ($A \leftarrow B \wedge C$) or arithmetic combination ($A \leftarrow B + C$). Thus a specification of the behavior, equivalent to the boolean equations of sequential circuits or to the differential equations of the circuit level, is a set of expressions (often called productions) which give the conditions under which such transfers will be made.

The fourth and last level in Figure 1 is called the PMS (Processor-Memory-Switch) level. At this level one sees only the most aggregate behavior of a computer system. The PMS level consists of central processors, core memories, tapes, disks, input/output processors, communications lines, printers, tape controllers, busses, teleprinters, scopes, etc. The computer system is viewed as processing a medium, information, which can be measured in bits (or digits, characters, words, etc.). Thus the components have capacities and flow rates as their operating characteristics.

The program level has been dropped from the original set of levels shown in Bell and Newell, because it was a functional level rather than a structural level.

Many notations are used at each of the four structural levels. Two of the less common ones are the PMS and ISP notations. A complete description of these notations is given in [Bell and Newell, 1971: Chapter 2]. Those aspects of PMS which are used in this book are described in Appendix 2. The ISP notation has evolved to ISPS and is described in Appendix 1.

VIEW 2: Levels of Interpreters

In contrast to the structural viewpoint given in View 1, this view is functional. According to this view, a computer system consists of layers of interpreters, much like the layers of an onion.

An interpreter is a processing system which is driven by instructions and operates upon state information. The basic interpretive loop, shown in Fig. 2, is most familiar at the machine language level, but also exists at several other levels.

To formalize the notion of levels of interpretation, one can represent a processing system by the diagram in Figure 3.

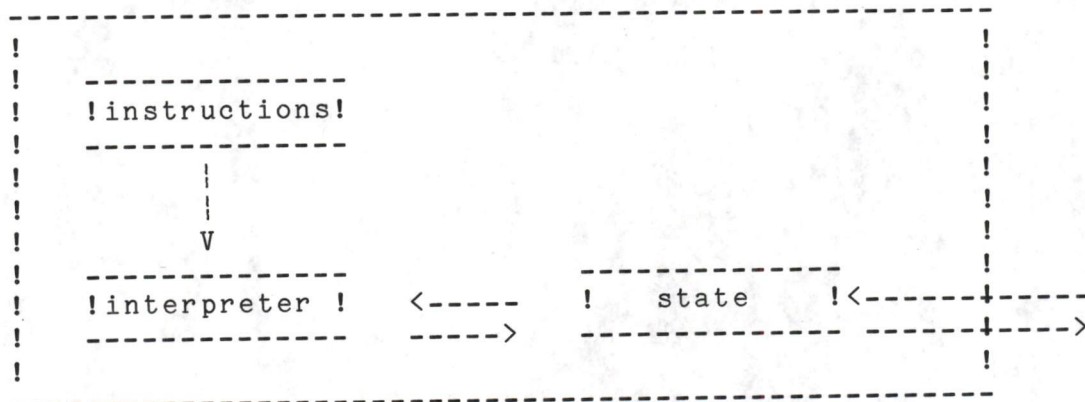


Figure 3: A processing system

The state information operated on by an interpreter is either "internal" or "external". This can best be understood by considering the "onion skin" levels of the five processing systems that form a typical airline reservation system. These levels are listed in Table 1.

Table 1

Level 4	Instruction:	seat allocation request message
	Interpreter:	airline reservation system
	Internal State:	number of requests pending at this moment location of passenger list on a disk file number of lines connected to system rotational position of disk
	External State:	number of reserved seats on a given flight airline name for a given flight
Level 3	Instructions:	Fortran statement codes
	Interpreter:	Fortran execution system
	Internal State:	memory management parameters user name main storage size location of disk files interrupt enable bits expression evaluation stack dimensions of arrays
	External State:	subroutine names values of data in arrays statement number program size value of an expression DO-loop variable value printed characters on line printer
Level 2	Instructions:	machine language instructions
	Interpreter:	processor
	Internal State:	program registers condition codes program counter
	External State:	data in main memory disk-controller registers
Level 1	Instructions:	microcode
	Interpreter:	micro machine
	Internal State:	instruction register flip-flops holding error status stack of microprogram subroutine links
	External State:	program registers

		condition codes
		program counter
Level 0	Instructions:	hardwired combinational network
	Interpreter:	sequential machine controlling the micro machine
	Internal State:	clock, counters, etc., controlling micro machine timing
	External State:	micro machine, console

The Level 0 system is the logic which sequences the Level 1 micro machine. The Level 1 system is a microprogrammed processor implemented in real hardware. It is the machine seen by the logic designer. The Level 2 system is the CPU. It is the machine seen by the machine language programmer. The Level 3 system shown here is a FORTRAN-language processing system. The Level 4 system is an airline-reservation system. These five systems form the hierarchy shown in Fig. 4, where each system is an interpreter which sequences through multiple steps in order to perform a single operation for the next-higher-level interpreter. The highest level system, the airline reservation system, is an interpreter operating on messages received from outside of the system. It tests and modifies states, and generates messages to send back outside the system, thus performing a single operation for the next-higher-level (outside) interpreter.

In practice, few systems are levels of pure interpreters, although layers are present. Deviations from the model have occurred for both hardware and software reasons.

In the hardware deviation case, the micro machine shown in Level 1 is often not present, but rather the Level 2 CPU is implemented directly using Level 0 sequential controllers. This practice of skipping Level 1 was initially due to

the lack of adequate read only memories, but is now generally limited to the
case of ultra high speed machines (Cray and Amdahl) ^{Rosach Corp. Corp.} which cannot tolerate the
fetch and execute cycle times associated with a control store.

There are two primary software driven departures from the pure interpreter model: (a) high-level-languages are usually executed by a compiler rather than by an interpreter, and (b) some layers are bypassed when more ideal primitives exist at deeper levels. Figure 5 illustrates this bypassing process. A pure interpreter implementation of Fortran would use an object time system (OTS) for all Fortran operations designated in the figure as "Type C". The object time system would require an operating system (OPSYS) for the interpretation of some of its operations, and the operating system in turn would be interpreted by the instruction set interpreter (ISP interpreter). However, the Type A operations in the figure would be directly interpreted by the instruction set interpreter.

Having presented the pure interpreter model, one can now return to the onion skin layered model and better understand how the different layers relate.

The macromachine hardware can be thought of as a base level interpreter. It is most often extended upward with an operating system. There may be several operating system levels so that the machine can be built up in an orderly fashion. A kernel machine might manage and diagnose the hardware components (disks, terminals) and provide synchronizing operations so that the multiple processes controlling the physical hardware can operate quasi-concurrently. Next, more complex operations like the file system and basic utilities are added, followed by policy elements such as facilities resource management and

accounting. As viewed through the operating system, one sees a much different machine than that provided by the basic instruction-set architecture. In fact, the resultant machine is hardly recognizable as the architecture most usually given by a symbolic assembler. It includes the basic machine, but has much more capable I/O and often the ability to be shared by many programs (or tasks).

Operating systems designers believe all these facilities are necessary in order to implement the next higher level interpreter--the standard language. The language level may include interpreters or compilers to translate back to the machine architecture for ALGOL, BASIC, COBOL, FORTRAN, etc. or any of the other hundred standard languages and their dialects.

Often an additional special language is used because an application can't be easily expressed using the standard language and it is necessary to have operations that are within the domain of the problem. Using a special language, various application subprograms can be created to enable specific application programs to be written. One then may build the application and finally the real user can get a problem solved in a cost-effective fashion provided there has been the right set of operations (languages) at each of the levels.

Sometimes the stratification is done to help manage complexity or to allow specialization of design activities. Sometimes the underlying layers are completely hidden from the user to aid ease of use.

Finally, note that using fundamentally different outer layers for a common

inner set of layers creates quite different machines, hence a set of onions.

Therefore it is important to realize that when dealing with common core hardware, multiple operating systems, languages and applications, a network of machines (a crop of onions) is formed, not just a single, layered machine.

In the final analysis, the number of levels is just another tradeoff.

Performance considerations lead to the deletion of levels, complexity leads to the addition of levels.

VIEW 3: Packaging Levels of Integration

This is a structural view that packages the various components (hardware and software) into levels. The levels for DEC computers in 1978 were as follows:

- 9 applications
- 8 applications components
- 7 special languages
- 6 standard languages
- 5 operating systems
- 4 cabinets (to hold complete hardware systems)
- 3 boxes
- 2 modules (printed circuit boards)
- 1 integrated circuits

This view is the most important in the book, because it shows how computer systems are actually structured, and hence how their costs are structured. Being a structural view of the object being sold, however, it is completely a function of the technology, the organization building the system, and the marketplace, all of which are so rapidly changing with time that the view could better be titled "Dynamic Levels of Integration". There are three major changes taking place:

1. Changes in the hardware levels, where the shrinking in physical size of functions has three effects:
 - a. Lower levels subsume higher levels
 - b. The semiconductor component supplier is forced to assume higher and higher level design responsibilities
 - c. Levels disappear

-
2. Changes in the software levels, again with three effects:
 - a. Each level grows in size as more functionality is added over time
 - b. More levels are added as minicomputers are applied to a broader range of applications
 - c. Functions migrate downward from level to level

 3. Changes in the hardware/software interface, where software functions migrate into hardware for higher performance

For the first of these areas of change, hardware levels, it is interesting to note that interconnection and packaging now constrain and limit design more than any other factor, excluding the basic lowest level component (semiconductor) technology.

The constraining and limiting by the interconnection and packaging take place because most manufacturing costs are associated with the physical structure. As interconnection levels must be introduced to build complex structures, many usually unwanted side-effects occur. The interconnection requires space and interferes with cooling air flow. Long interconnections increase signal transmission delays, and these reduce performance. Signal transmission not only makes the computer susceptible to electromechanical interference, but also may radiate electromagnetic waves will need to be controlled. Finally, there is the domain of industrial design or aesthetics, and everyone is a self-based authority on appearance.

Figure 6 shows the costs of various levels-of-integration versus time for small computers. The cost depends partly on implementation and architecture word-length. As the word-length is made shorter, there is some economy particularly for very small computers, because some levels of integration cease to exist. For example, most hand held calculators are implemented using 4-bit, stored program computers with fixed programs that occupy a single IC. They have no associated modules, backplanes, boxes and cabinets--only a single package that fits one's hand.

Note from Fig. 6 that semiconductors, the lowest level of technology, have had the greatest price decline. Modules have a lesser price decline because they are a mix of ICs, printed circuit boards, component insertion labor, and testing labor. The price decline for the IC portion of the module cost is moderated by the labor intensive nature of module fabrication, thus producing a price decline for modules that is markedly less than that for ICs. At the box level-of-integration, power supplies and metal or plastic boxes are also labor intensive and further moderate the price decline provided by the IC's. Finally, as boxes are integrated (by people), and applied at a system level (by people), the price decline has almost disappeared.

Many of the cost improvements brought about by new technology are derivative. They are by-products of using less power and less space, thus avoiding the labor intensive levels of packaging integration.

An astute marketing-oriented person might ask, "How, with all the technology, can we do something unique so that we can maximize the benefit from the technology without having to pay so much for labor intensive items such as

packaging?" One answer: "Reduce prices by not providing a power supply and mounting hardware. Let the user provide all added-on parts and mount the computer as needed. In this way, the price, though not necessarily the total cost to the user, is reduced. We'll sell at the board level." Computer Automation followed this philosophy when they introduced the "Naked Mini" TM so that users could supply more added value - packaging and power technology.

A similar effect can be seen in the PDP-11 series since the PDP-11/20's introduction in 1970. At that time, the 4,096 word PDP-11/20 (mounted in a box) sold for \$9300. In 1976, the boxed version of an LSI11 cost \$1995, reflecting a factor of 4.7 improvement over the PDP-11/20. The 4,096 word core memory module used in the 11/20 sold for \$3500, while a 16,384 word MOS memory module for an LSI11 sold for \$1800, reflecting a factor of 7.8 improvement.

The changing levels of integration have also changed the domain of the semiconductor suppliers. In the early 70s, Intel, North American Rockwell, and other semiconductor companies began to use the higher semiconductor densities to reduce the number of levels of integration by packaging a complete processor-on-a-chip. These organizations had assimilated logic design, but were frustrated because their customers could really not identify higher functionality units (beyond memory) requiring on the order of 1,000 gates on a chip. Also, the speed of these high density units was quite low.

They discovered that the best finite state machine to make was just a simple computer, because it provided the finite state machine plus the useful functions that were not covered by switching circuit theory. It became "simply

a small matter of programming" to do something useful. While programs for these simple computers cost \$1-\$100 per instruction to write, the prices for processors-on-a-chip have followed a very steep decline of up to 50% price reduction/year.

Robert Noyce, of Intel, presented Fig. 7 in October 1975. It illustrates what has been happening in the semiconductor industry, and has been modified slightly to show the technology that DEC has assimilated with time. It indicates the breadth semiconductor manufacturers now have in technology, starting from semiconductor device level, through the view Noyce has of the various levels-of-integration, and continuing into end user applications.

The levels-of-integration viewpoint can be summarized as components of one level being combined into a system at the next highest level in a hierarchy. A level denotes that there is a single conceptual design discipline or set of interacting disciplines which determine the function, structure, performance and cost of the constituent level. "Level" is a deceptive word, because as Fig. 8 shows, the structure is actually a lattice or network, and not a tree. Furthermore, each level can be nested itself.

VIEW 4: Computer Classes: A Marketplace View

Because it is the complete marketplace process that produces the computer, this view is the most complex. A computer is characterized (in terms of marketability) as a function of price, performance, and time of introduction in what might appear to be a commodity-like environment.

Because various computers operate at different performance rates and at various costs, computation can be purchased in multiple ways, and price/performance ratios will thus affect marketability. For example: computation can be supplied by a shared large, central batch computer; each organizational entity can own and operate a shared minicomputer; an individual can operate a single desk top system; or each individual can operate a programmable calculator.

Price/performance is not the sole factor determining marketability, however. Program compatibility with previous machines is very important. When users write programs, there is a lifetime associated with the use of those programs, and thus a need to have compatible processors for running those programs today and for a substantial time into the future. Thus, while the supply of rapidly evolving technology permits new designs to be more cost effective, and even radical, continuity with the past must exist.

The Marketplace View begins with a description of how technology provides basic improvements with each new generation at approximately six year intervals. An example of alternative new designs is given to show why most new designs usually provide increased performance at constant price. With this

background, the four price/performance classes that have evolved over the last four generations are presented.

The influence of technology on the computers that are built and taken to the marketplace is so strong that the four generations of computers have been named after the technology of their components: vacuum-tubes, transistors, integrated-circuits (multiple transistors packaged together), and LSI circuits (Large-Scale Integration). Every electronic technology has its own set of characteristics (e.g., cost, speed, heat dissipation, packing density, reliability), all of which the designer must balance. These factors combine to limit the applicability of any one technology; typically, one technology is used until a limit is reached, or another technology supersedes the old.

Design Alternatives

When an improved basic technology becomes available to a computer designer, there are four paths the designs can take to incorporate the technology:

1. use the newer technology to build a cheaper system with the same performance;
2. hold the price constant and use the technological improvement to get an increase in performance;
3. push the design to the limits of the new technology, thereby increasing both performance and price; or

-
4. find a drastically new structure using the computer as a basic archetype (e.g., calculators) such that the design can be considered off the evolutionary path.

Figure 9 shows the trajectory of the first three of the design alternatives. In general, the design alternatives occur in an evolutionary fashion as in Fig. 10 with a first (base) design, and subsequent designs evolving from the base.

In the first design style, the performance is held constant and the improved technology is used to build lower price machines which attract new applications. This design style has as its most important consequence the concept of the "minimal computer". The minimal computer has traditionally been the vehicle for entering new applications, since it is the smallest computer that can be constructed with a given technology. Each year, as the price of the minimal computer price declines, new applications become economically feasible.

The second, constant cost alternative uses the improved technology to get better performance, and will usually yield the best increase in total system-cost-effectiveness. This approach provides a growth in performance and quality at a constant price and is probably the best for the majority of existing users.

The third alternative is to use the new technology to build the most powerful machine possible. New designs using this alternative often solve previously unsolved problems, and in doing so advance the state of the art. This design

alternative must be used cautiously, however, because going too far in price or performance, (i.e., building beyond the technology) is dangerous and can lead to a zero performance, high cost product. There are usually two motivations for operating at this leading edge: preliminary research motivated by the knowledge that the technology will catch up; and national defense, where an essentially infinite amount of money is available because the benefit--avoiding annihilation--is infinite.

Table 2 shows the effect of pursuing the two design strategies of 1) constant performance at decreased price, or 2) constant price at increased performance.

Table 2: Using New Technology for Constant Price and Constant Performance Designs

Introduction time (generation)	t	t+1	t+1	t+1
Design style	base case	constant price, increased performance	constant performance, decreased price	constant performance, decreased price
Application	base	base	base	new base
Computer price	1	1	0.5	0.5
Operating costs (range)	2-4	2-4	2-4	1-2
Total cost	3-5	3-5	2.5-4.5	1.5-2.5
Performance (and improvement)	1	2	1	1
Improvement (in total cost)	1	1	.83-.9	.5-.6
Performance/price (computer only and improvement)	1	2	2	2
Performance/Total-cost	.33-.2	.66-.4	.4-.22	.66-.4
Improvement in (Perf./total cost)	1	2	1.21-1.1	2

The first column gives the base case at a given time, t. Since this is the base case, the price, performance and performance/price ratio of the computer are all 1. As the computer is applied to a particular environment, operational overhead is added at a cost of 2 to 4 times the original price of the computer; the total cost to operate the computer becomes 3 to 5, and the performance/total-cost ratio is reduced to between .33 and .2 (depending on the total cost).

Now assume the same operating environment, with the same fixed (overhead)

costs to operate, at a new time, $t+1$, when technology has "improved" by a factor 2. Two alternative designs are carried out, one is at constant price/higher performance and the other is at constant performance/lower price (cols. 2 and 3). The application is constant in three cases (cols. 1-3) and a new application is discovered for the fourth case (col. 4). Both the constant cost and constant performance designs give the same basic performance/cost improvement--when only the cost of the computer is considered. However, when one considers the high fixed overhead costs associated with the application (cols. 1-3), there is a relatively small improvement in cost performance/cost, although there has been a cost savings of 17 to 10 per cent. The greatest gains come in applying the computer with greater performance and getting the attendant factor of 2 gain in performance and in performance/price ratio.

To summarize, the constant-price / increased-performance design style gives a better gain because operating costs remain the same. Its gain can only be equalled by the constant-performance design style when operating costs are halved upon its application. This only occurs when a new application is tackled, such as that shown in column 4.

Computer Classes

Applying the three design styles shown in Fig. 9 over several generations produces the plot given in Fig. 11. These figures lead to one of the most interesting results of the Marketplace View, which is that computer classes can be distinguished by price and named as follows: sub-micro (to come in the next generation--say by 1980), micro, mini, midi, maxi, and super. The classes midi- and maxi- are sometimes referred to by the single,

non-descriptive name, mainframe.

When one distinguishes computer classes by price, a new range of price can be made possible by new technology and create a new class. The new class appears at the low end of the price scale where the minimal computer is introduced at a significantly lower price level than existing computers.

While the measure used to define a new class is price, the measure defining an established class is performance. This is because once a new class has become established in the marketplace, the users become familiar with what computers of that class can do for their applications, and tend to characterize that class on a performance basis. The characterization of existing classes on a performance basis is important to this discussion because at each new technology time, performance increases by one category, and midi performance becomes available on a mini, for example.

The effect of technology upon computer classes can be summarized in the following thesis:

Continual application of technology via the two major design styles results in

- (a) price declines which create new classes of computers,
- (b) new classes becoming established classes, and
- (c) established classes becoming encroached upon.

To refine the thesis, some additional commentary is in order. First, some

question may arise as to how much of a price reduction is necessary to create a new class. The continuity implied by the thesis is deceptive in that it suggests that new classes come about by the continual application of the constant-performance, decreasing cost style of design. Viewing the industry as a whole, this is true. However, a new class is usually not created by the same organization that is designing computers in existing classes. A new company, or new organization within a company, is usually required to provide the requisite fresh viewpoint needed to create a new class. It is the fresh viewpoint and not some arbitrary amount of price reduction that creates a new class.

The minicomputer class came about by DEC founders taking the view that simple machines, unencumbered by the costly general purpose facilities of the mainframes (many data types, sharing mechanisms, etc.), could be built and would be viable.

While the minimal design philosophy pervaded for nearly 10 years as the minicomputer, the semiconductor-based, processor-on-a-chip computer (called microcomputers) name identifies a significantly lower priced computer. This lower cost is fundamentally based on the elimination of a level of integration, interconnection. The microcomputer is dispelling the lethargy growing around the constant-price minicomputer of the mid-1970's and is generally being promoted by companies not in the minicomputer business, or by separate organizations within the minicomputer companies.

The sub-microcomputer is predicted to be a complete computer (with both processor and program memory) on a single chip, although it is not as radical

as the microprocessor which eliminated the concept of the box-of-boards computer.

In both the mini and micro cases a fresh organization broke out. Why is a fresh viewpoint needed? Because existing organizations, like most human organizations, act to preserve the status quo, and produce constant-price computers. This can be explained by examining each part of the organization in turn.

1. The existing user, who is the potential customer for a new system has a set of fixed costs (overhead). Thus by the same arguments that were just discussed in conjunction with constant cost and constant performance design styles, the existing user generally wants more performance rather than lower price. Productivity improvements are required to stay at equilibrium with his organization and to sustain or increase funding. Operating costs (personnel, paper, power, etc.) are likely increasing due to inflation, and although a lesser priced machine could hold performance/cost constant, there would be no productivity gain. Requiring less or equal funds, would ultimately cause the loss of organizational power since this is based on people, expenses, ability to provide increased service, etc. Thus, the pressure is to get larger, more powerful machines. Furthermore, his software investment requires that the new system be compatible with his earlier system.
2. The sales force respond mainly to orders which are lost because of poor performance, because its measure is so tangible. Price (and even

lack of functionality) is considered only when the performance of two alternatives is equal. The amount of work a salesperson has to do is measured in sales yields and, indirectly, the number of systems sold. The reduction of the sales price of a given system by a factor of two, requires selling twice the number of systems to bring in a constant amount of sales dollars. This implies working significantly harder (neglecting the effects of price elasticity). For example, within DEC, because of the wide price range of its products, a salesperson need sell only one large system per year, while at the low end he must sell 1500.

3. Marketing reinforces the inputs from the sales force, while interjecting its own bias and noise. Because of the broad range of applications, it presents conflicting product definitions. Thus, a new product must have: the performance of the current most competitive machines; the price of the most marginal competitor (who may be losing money to enter the market); the software of all the competitors (since each differentiates itself by unique software); and all the service of the largest (and possibly highest price) competitor.
4. Engineering. The new product specification input is to improve performance as seen in paragraph 1 above. Technology pushes engineers to use higher-performance components since technology suppliers behave in an identical fashion to the computer supplier. Although there are various styles of designs (and designers), the most common tendency of the designer is to use new technology to provide higher performance,

and to solve the problems inherent in subsequent designs rather than working to reduce costs again. The PDP-8 family provides an example.

5. Manufacturing constraints and planning are similar to sales. A plant is measured in units, quality level, and unit cost. Preserving a given cash flow, when faced with radically different cost products, usually requires substantially different planning, materials handling, manufacturing, testing, etc. The radical changes for a higher volume business (i.e., long term planning) is difficult. Product design change ideas fed back to engineering usually address solving previous problems that add to product cost, rather than addressing radical concepts that could create a new computer class.

The new organization, in addition to providing a fresh viewpoint, must also provide the capability and willingness to deal with progress in multiple areas of technology if the maximum evolution rate for the computer price class thesis is to be maintained. The computer class thesis may seem to suggest that there is one monolithic technology, steadfastly progressing, which provides incremental improvements. In fact, there are several technologies. For example, to fully exploit each new circuit technology improvement, a matching advance in packaging technology is needed. Higher circuit densities demand new cooling methods; lower-cost module carriers are needed to match the lower cost microprocessor-based control techniques. However, since engineering projects tend to minimize the number of risk dimensions, a new circuit technology is generally used twice: once with existing packaging and a second time with packaging that matches the technology. There are examples in both the PDP-11 and S/370 families. In addition to packaging technology, mass storage, primary

memory, I/O units, logic, and power are areas whose technology must move ahead in step. Unfortunately, in practice this does not occur. Memory prices have consistently halved every two years, whereas electro-mechanical technology is slowly increasing in price.

One of the by-products of the use of new technology is that conflicts occur within the established computer classes. An established computer class, since it is defined on the basis of performance, is entered by constant cost/higher performance successors from the class below it. Moreover, suppliers within a class are, by their dominant constant price/ higher performance evolution, operating to move up out of their class.

While movement by computer designs and computer suppliers between and amongst the various classes may be encouraged by price and performance trends, the speed with which that movement occurs is moderated by software compatibility considerations. The computer class thesis is not meant to imply that each class implements the same instruction set and PMS-level configurations, and that they differ only in speed. Rather, much specialization occurs in each class and many of the attributes of the higher performance machines appear to substantially lesser degrees in the lower performance classes. For example, there are more data types in the larger machines, their address spaces (both physical and virtual) are larger, and the software support is generally broader. Resources devoted to increasing reliability and availability are more common in the higher priced machines. The PDP-11 family, from the VAX-11/780 down to the LSI-11, exemplify these functionality differences.

Definition of the Minicomputer

The concept of computer classes that can be distinguished by price and named sub-micro, micro, mini, midi, maxi, and super, may be of assistance in finding a definition for the minicomputer, a definition which has thusfar been rather cloudy.

While the above list of classes suggests that minicomputers are those computers whose prices fall between microcomputers and midicomputers, and thus somewhere near the middle of the range of computers available, earlier definitions use the term mini to denote minimal. For example [Bell, 1971]:

"Minicomputers (for minimal computers) are a state of mind (or the designers' minds); the current logic technology, and the characteristics found in larger computers are combined into a package which has the smallest cost."

Iann Baron [Computing Europe, December 1975] defined the minicomputer in somewhat similar terms:

"Minicomputers are a state of mind. It is not useful to define them in terms of price or capability because of the rapid changes in technology."

The Marketplace View defines new computer classes according to price and established computer classes according to performance. This would suggest that a definition of the minicomputer should include some historical data on price and some comments on performance, or at least give some indications of performance by discussing applications and configurations. In 1977, Gordon Bell provided such a hybrid definition for the Director of Computer Resources, U. S. Air Force. The definition was as follows:

"MINICOMPUTER: A computer originating in the early 1960's and predicated on being the lowest (minimum) priced computer built with current technology. From this origin, at prices ranging from 50 to 100

thousand dollars, the computer has evolved both at a price reduction rate of 20% per year and has also evolved to have increased functionality and a slightly higher price with increasing functionality and performance.

Minicomputers are integrated into systems requiring direct human and process interaction on a dedicated basis (versus being configured with a structure to solve a wide set of problems on a highly general basis).

Minicomputers are produced and distributed in a variety of ways and levels of integration from: printed circuit boards containing the electronics; to boxes which hold the processor, primary memory, and interfaces to other equipment; to complete systems with peripherals oriented to solving a particular application(s) problem. The price range(s) for the above levels-of-integration, in 1978, are roughly: 500-2,000; 2,000 to 50,000; and 5,000 to 250,000."

Economy of Scale

Another important aspect of the Marketplace View, in addition to the idea of dividing computers into classes based on price for new classes and performance for existing classes, is the concept of economy of scale in the computer industry.

For nearly all man-made objects, such as transportation vehicles, electricity generators, or buildings, there is usually some economy of scale because there are high fixed costs that do not increase as rapidly as the output of an object increases.

For computers, factors leading to economies of scale often apply over several dimensions. The same software can be used on many models. Sales and field service people can attend to a wide range of equipment. Manufacturing facilities can be adapted to produce different models.

Grosch (1953) suggested that there was an economy of scale for computers according to the performance/price relationship:

2

Performance = constant x Price

Several studies [Bell and Newell, 1971; Knight, 1966; Solomon, 1966; Phister, 1976; Sharpe, 1969; Turn, 1974] have examined whether this is true for a given set of machines. Over a narrow range of prices studied (a factor of four), a linear approximation to the data appeared to fit nearly as well as the square law did. See Fig. 12. While a square law performance/price relationship is thus probably too extreme to apply to all computer system components, there are some cases where a square law does apply, and other cases where substantially nonlinear performance/price relationships do exist. In short, the square law approximation, while not perfect, is a useful model.

A computer component that could be predicated on a nearly square law relationship is the core memory. The electronic selection is square law; a doubling of the selection circuitry provides access to a four times larger stack. The manufacturing cost for larger stacks follows a less dramatic economy of scale since there is a high set-up cost to threading core memories, and larger memories spread this cost over more bits. All other costs are roughly linear, including such overhead costs as memory packaging, memory power, and memory interface.

Software Compatibility

Equally important to the hardware considerations of the Marketplace View are the software considerations. Nearly all modern computer designs are part of a compatible computer family which extends over price and time. Compatibility considerations are based on the economic necessity to utilize a common software base. The users' investment in software dwarfs that of the manufacturer, assuming the machine is successful. For example, if there is only one man year of software investment associated with the 50,000 PDP-11s, and each man year costs about \$40,000 and produces something on the order of 5,000 instructions, then there is a cumulative investment of 2 billion dollars and 250 million lines of program for the PDP-11. This investment is roughly the same scale as the original hardware cost.

Since technology provides such rapid changes over the generations, it is obvious that there must be backward (in time) compatibility in order to build on and preserve the user's program base. He must be able to operate programs unchanged to take advantage of improvements brought about by technology changes.

In a similar way, compatibility over a range of machines, at a given time is desirable so a user may select a machine that matches his problem set, while having the comfort that the problems can change and there will be a sufficiently large or small machine available to use in dealing with the new problems.

Thus the goals for a complete, compatible computer product line (i.e., set of products) might be:

-
1. the widest range of products in terms of price offering;
 2. the largest economy of scale factor for performance that separates the models;
 3. the smallest number of models to minimize costs of design, manufacture, selling and spares inventory;
 4. the most cost-effective/competitive products (nearest to competitive machines) for each price in the space; and
 5. backward compatibility with past machines.

Goals 3 and 4 are contradictory. How can the number of models be minimized, while at the same time providing competitive machines over a continuous space? A possible method is described in Bell and Newell (1971) where a multiprocessor alternative to the System/360 is proposed. This approach provides increased reliability and the ability to easily upgrade performance with minor additions.

Chapter 15 "The Evolution of the PDP-11" discusses quantitatively the performance range spanned by the PDP-11, compares it with the span of the S/360, and contrasts the different techniques used.

This discussion of the Marketplace View has been a qualitative explanation of the effect of technology on the computer industry. It is an engineering view, rather than one that would be given by technology historians or economists. The twenty years described in this book and the individual cost and

performance measures surely invite analysis by professionals. The studies reported in [Sharpe, 1969] are a good departure point.

VIEW 5: Computer Classes: an Applications/Functional View

Because of the general-purpose nature of computers, all of the functional specialization occurs at the time of programming rather than at the time of their design. As a result, there is remarkably little shaping of computer structure to fit the function to be performed.

The shaping that does take place uses five primary techniques.

1. PMS-level configuration

A configuration is chosen to match the function to be performed. The user (designer) chooses the amount of primary memory, the number and types of secondary memory, the types of switches, and the number and types of transducers to suit his particular application.

2. Physical packaging

Special environmental packaging is used to specialize a computer system for certain environments, such as factory floor, submarine, or aerospace applications.

3. Data-type emphasis

Computers are designed with data types (and operations to match) that are appropriate to their tasks. Some emphasize floating-point arithmetic, others string handling. Special-purpose processors, such as Fast Fourier Transform processors, belong in this category also.

4. Operating-system

The generality of the computer is used to program operating systems that emphasize batch, time-sharing, real-time, or transacting processing needs.

Current Dimensions of Use

In the early days of computers, there were just two classifications of computer use, scientific and commercial. By the early 1970's, computer use had diversified to seven different functional segmentations: scientific, business, control, communication, file control, terminal, and time-sharing. Since that time, very little has changed in terms of functional characterization, but two points are worthy of comment. First, file control computers still have not materialized as mainstream separate functional entities, despite isolated cases such as the IBM 3850 mass-storage system, and second, terminal computers have evolved to a much higher degree than expected.

The high degree of evolution in terminals has been due to the use of microprocessors as control elements, thus providing every terminal with a stored program computer. Given this generality, it is has been an easy matter to provide the terminal user with facilities to write programs. In turn, this phenomenon has affected the evolution of time-sharing (when using the term to denote close man-machine interaction as opposed to shared use of an expensive resource).

Functional segmentation into categories with labels such as "business", "control", communication", and "file control" reflects a naming convention rooted in the old two category "scientific"/"commercial" tradition. An alternative classification, more useful today, is the segmentation scheme

 shown in Table 3. It is based on the intellectual disciplines and environment (e.g., home based) using and developing the computer systems. It shows the evolving structures in each of the disciplines, permitting one to see that nearly all the environments evolve to provide some form of direct, interactive use in a multiprogrammed environment. The structures that interconnect to mechanical processes are predominately for manufacturing control. Other environments, such as transportation, are also basically real time control. Another feature of discipline-based functional segmentation is the fact that each of the disciplines operate on different symbols. For example, commercial (or financial control) based environments hold records of identifier names for entities (e.g., part number) and numbers which are values for the entity (e.g., cost, number in inventory).

Table 3: Discipline/Environment Based Functional Segmentation Scheme

Commercial environment

financial control for industry, retail/wholesale, and distribution
 billing, inventory, payroll, accounts receivable/payable
 records storage and processing
 traditional batch transaction
 processing against data base business
 analysis (includes calculators)*

Scientific, engineering and design based environment

numbers, algorithms, symbols, text, graphs storage and processing
 traditional batch computation*
 data acquisition*
 interactive problem solving*
 real time (includes calculators and text processing)
 signal and image processing*
 data base (notebooks and records)

Manufacturing environment

record storage and processing
 batch*
 data logging and alarm checking
 continuous real time control
 discrete real time control

machine based
people/parts flow

Communications and Message Based

message switching
front end processing
store and forward networks
speech input/output
terminals and systems
word processing, including computer conferencing and publishing

Transportation systems

network flow control
on-board control

Education-based

computer assisted instruction
algorithms, symbols, text storage and processing
drill and practice
library storage

Home-based using TV set

entertainment, record keeping, instruction, data base access

*Implies continuous program development

The scientific, engineering, and design-based disciplines use various algorithms for deriving symbols or evaluating values. Text, graphs, and diagrams are the major ways of representing objects, and have to be processed. For these environments, the computer has changed from a calculator (it was initially funded to do trajectory calculations for ballistic weapons) to a sophisticated notebook for keeping specifications, designs and scientific records. Whereas the computer was initially only used as a transducer to collect data to be analyzed later on larger machines, it has since evolved to direct recording and analysis of time varying signals and images and even to direct analysis and control. Many transducers now have, or will have, computers embedded in them in order to encode information at a high level so that their output does not have to be processed by another computer.

Connections to other larger computers are instead used solely in a network fashion to handle graphical display and control functions.

The function of computers in both the manufacturing environment and the commercial environments has evolved from simple record keeping to direct on-line human control.

Process-control computers have evolved from their initial use as assisting human operators (controllers) with data logging and alarm condition monitoring to full control of processes with either human or second computer backup. The structure of the computer and the control task varies widely depending on whether it be a continuous process (e.g., refinery, rolling mill) or a discrete process (e.g., warehouse, automotive, appliance manufacturing).

Transportation applications for aircraft, trains, and eventually automotive vehicles, are a form of real time control that use both discrete and continuous control. Control is carried out in two parts: on board the vehicle and the network (airspace, highway) that carries the vehicles. The transportation control function dictates three unique characteristics for the computer structure:

1. very high reliability. Society has placed such a high value on a single human life that all computers in this environment can not appreciably raise the likelihood of a fatality.
2. very small size for on-board computers.
3. extreme operating and storage temperature range for on-board computers - especially for automotive vehicles.

Communications and message based computers have evolved from telephone switching control, message switching, and front ends to other computers to be

the dominant part of communications systems. With these evolving systems, the communications links have changed from analog-based transmission to sampled-data, digital transmission. By using all-digital transmission, data and voice (and video) can ultimately be used in the same system. Voice transducers enable speech communications with the computer.

Word processing (i.e., creation, editing, and reproduction) together with the long term storage and retrieval, and transmission to other sites (i.e., electronic mail) have evolved from several systems:

1. Conventional torn-tape message switching (e.g., TWX, Western Union, Telex).
2. Terminals with local storage and editing (i.e., Flexowriters (R), Teletype (R) ASR's, magnetic card/magnetic tape automatic typewriters, and the evolving stand-alone word processing terminals.
3. Large shared text preparation systems for centralized documentation preparation, newspaper publication, etc.
4. Large systems with central filing and transmission (distribution). These will negate need for substantial hard copy. With these systems, text can be prepared either centrally with the system,, or with local intelligent word processing systems.
5. Computer conferencing. People can sit at terminals and converse with others without leaving their office.

The education-based environment implies a system which is a combination of transaction processing (for the human interaction part), scientific computation as the computer is required to simulate real world conditions (i.e., physical/natural phenomena), and information retrieval from a data base. These systems are evolving from the simple drill-and-practice systems which use a small simple algorithm, through simulation of particular real-world phenomena, to knowledge-based systems which have a limited, but useful, natural-language-communications capability.

Home based computers are beginning to emerge. The dominant use to date is in providing entertainment in the form of games that model simple, real-world phenomena, such as ping-pong. Appliances are beginning to have embedded computers that have particular knowledge of their environments. For example, computer-controlled ranges can cook particular food in fairly standard ways. Alternatively, cooking can be controlled by embedded temperature sensors. Simple calculators to record checkbooks have existed for quite some time. These will soon evolve to provide written transactions for recording and control purpose. Many domestic activities are in essence scaled-down versions of commercial, scientific, education, and message environments.

With the evolution of each machine class (super, mainframe, minicomputer, microcomputer, and the hand-held calculator), one can see several cases of machine structures which begin as highly specialized and evolve to being quite general. This evolution is driven by applications in accordance with the Applications/Functional View of computer classes.

The applications-driven evolution toward generality applies to both hardware and software. As a hardware example, consider the case of a computer installations using large, highly general computers, where minicomputers are applied to offload the large computers. The first application of the minicomputer is thus on a well-defined problem, but then more problems are added, and the minicomputer system is soon performing as a general computation facility, with the help of a general-purpose operating system. A similar effect can be in software, where operating systems take on multiple functions as they evolve with time because users specify additional needs, and operating systems designers like to add function. Thus a COBOL run-time environment

might be added to a simple FORTRAN-based real-time operating system. At the next stage a comprehensive file system might be added. In the hardware system, the next step in the evolution is usually offloading the minicomputer, and in the software case, the next step is often the development of a new small, simple, and fast operating system.

Part of this evolution is due to the inherent generality of a computer, and part is a consequence of constant-cost design philosophy. The evolution is observable in computers of all classes, including calculators. The early scientific calculators evolved from just having logs, exponentials and transcendental functions to include statistical analysis, curve fitting, vectors and matrices.

Machines, then, evolve to carry out more and more functions. Since a prime discriminant is data type, Figure 13 is presented to show an estimate of data type usage for various applications, using mostly high-level data types, e.g., process descriptions. The estimates shown are very rough, because attempts to measure such distributions to date have not shown marked differences across applications (except for numerical vs non-numerical) because the data types have not been of a sufficiently high level.

VIEW 6: The Practice of Design

Whereas previous views emphasized the object being designed, this is a view of the design process which gives rise to the object. Two models of design, those of Asimow and Simon, are presented, followed by some remarks on factors that particularly influence computer design.

In Introduction to Design [1962], Asimow gives a general perspective of engineering design and how the formal alternative generators and evaluating procedures are used. He also indicates where these formalisms break down and where they don't apply. He defines engineering design as "a purposeful activity directed toward the goal of fulfilling human needs, particularly those which can be met by the technological factors of our culture."

Asimow distinguishes two types of design: design by evolution and design by innovation. While there are examples of both in this book, design by evolution predominates both in this book and in the computer industry.

Asimow's first diagram (Fig. 14), called Philosophy of Design, shows the basic design process. Asimow lists the following principles [Asimow, 1962: 5-6].

1. **Need.** Design must be a response to individual or social needs which can be satisfied by the technological factors of culture.
2. **Physical realizability.** The object of a design is a material good or service which must be physically realizable.
3. **Economic worthwhileness.** The good or service, described by a design, must have a utility to the consumer that equals or exceeds the sum of the proper costs of making it available to him.
4. **Financial feasibility.** The operations of designing, producing, and distributing the good must be financially supportable.

-
5. **Optimality.** The choice of a design concept must be optimal among the available alternatives; the selection of a manifestation of the chosen design concept must be optimal among all permissible manifestations.
 6. **Design criterion.** Optimality must be established relative to a design criterion which represents the designer's compromise among possibly conflicting value judgments that include those of the consumer, the producer, the distributor, and his own.
 7. **Morphology.** Design is a progression from the abstract to the concrete. (This gives a vertical structure to a design project.)
 8. **Design process.** Design is an iterative problem-solving process. (This gives a horizontal structure to each design step.)
 9. **Subproblems.** In attending to the solution of a design problem, there is uncovered a substratum of subproblems; the solution of the original problem is dependent on the solution of the subproblem.
 10. **Reduction of uncertainty.** Design is a processing of information that results in a transition from uncertainty about the success or failure of a design toward certainty.
 11. **Economic worth of evidence.** Information and its processing has a cost which must be balanced by the worth of the evidence bearing on the success or failure of the design.
 12. **Bases for decision.** A design project (or subproject) is terminated whenever confidence in its failure is sufficient to warrant its abandonment, or is continued when confidence in an available design solution is high enough to warrant the commitment of resources necessary for the next phase.
 13. **Minimum commitment.** In the solution of a design problem at any stage of the process, commitments which will fix future design decisions must not be made beyond what is necessary to execute the immediate solution. This will allow the maximum freedom in finding solutions to subproblems at the lower levels of design.
 14. **Communication.** A design is a description of an object and a prescription for its production; therefore, it will have existence to the extent that it is expressed in the available modes of communication.

Asimow goes on to define the phases of a complete project.

1. **Feasibility study.** The purpose is to determine some useful solutions to the design problem. It also allows the problem to be fully defined and

tests whether the original need which initiated the process can be realized. Here the general design principles are formulated and tested.

2. **Preliminary design.** This is the sifting, from all possible alternatives, to find a useful alternative on which the detailed design is based.
3. **Detailed design.** This furnishes the engineering description of a tested and producible design.

While the above are the primary design phases, there are four succeeding phases resulting from the need for production and consumption by the outside world.

4. **Planning the production process.** This is really another design process which is simply a special case of design. The goal is to design and build the system that will produce the object.
5. **Planning for distribution.** This activity includes all aspects related to sales, shipping, warehousing, promotion, and display of the product.
6. **Planning for consumption.** This includes maintenance, reliability, safety, use, aesthetics, operational economy, and the base for enhancements to extend the product life.
7. **Retirement of the product.**

Obviously all of these activities overlap each other in time, and interact as

the basic design is carried out. Phister (1974) posits a model of this process (Figs. 15 and 16) and gives the amount of time spend in each activity (Fig. 17) for a hardware product.

A more abstract model of design is the one that Simon uses for human problem solving and calls "generate and test". In The Sciences of the Artificial, Simon [1969] discusses the science of design and breaks the problem into: representing the design problem alternatives, the search (i.e., generating alternatives), and computing the optimum. When it is too expensive to search for the optimum, as is often the case, one resorts to selecting satisfactory alternatives (which Simon calls "satisficing" alternatives) and testing them. For most parts of computer design, the design variables are selected on the basis of satisfactory rather than optimum choice. Simon also discusses the tools of design, including the use of simulation as an alternative to building the complete system, and the use of simulation as a method of evaluating the behavior of various alternatives.

In addition to the contribution of "generate and test" to the Practice of Design View presently being discussed, Simon's work has also contributed indirectly to views discussed earlier in the chapter. In his discussion of the importance of the design hierarchy, Simon introduced the notion of "architecture of complexity". The first three views of this chapter have been influenced by this hierarchies view.

The search for design optimums, whether it be by "generate and test" or some other alogrithm, often encounters the problem of design representation. The more representations one has, the larger is the number of design problems that

can be tackled, and hence the closer one can get to a global optimum. Most disciplines have at least two: schematic and visual representations. In chemical engineering, heat balance is obtained by thermodynamic equations, not from a plant piping diagram. In the design of power supplies, transformer design is accomplished using equivalent circuits, not by using physical representations. In the design of computer buses, most designers work with timing diagrams, although state diagrams and Petri nets are alternative representations.

In general, the importance of alternative representations in computer engineering is not well understood. The large number of representations that do exist at the programming level is deceptive. There are many different algorithmic languages, but they differ mostly in syntax, not in semantics. It is too simplistic to think that computer design should be a well-defined activity in which mathematical programming can be employed to obtain optimum solution. There are major problems, five of which are listed below:

1. the cost function is multi-variable
2. the primary measure, performance, is not well understood
3. the objective function that relates cost and performance is not understood
4. objectives are not as objective as they look
5. there is a dynamic aspect (because the technology changes rapidly) which is hard to quantify.

These problems are explored in the following extract from a discussion of design given in [Bell, Grason, and Newell, 1972: 23-24].

Objectives can often be stated as maximizing or minimizing some measure on a system. A system should be as reliable as possible, as

cheap as possible, as small as possible, as fast as possible, as general as possible, as simple as possible, as easy to construct and debug as possible, as easy to maintain as possible -- and so on, if there are any system virtues that have been left out.

There are two deficiencies with such an enumeration. First, one cannot, in general, maximize all these aspects at once. The fastest system is not the cheapest system. Neither is the most reliable. The most general system is not the simplest. The easiest to construct is not the smallest, and so on. Thus, the objectives for a system must be traded off against each other. More of one is less of another and one must decide which of all these desirables one wants most and to what degree.

The second deficiency is that each of these objectives is not so objective as it looks. Each must be measured, and for complex systems there is no single satisfactory measurement. Even for something as standardized as costs there are difficulties. Is it the cost of the materials -- the components? Does one use a listed retail cost or a negotiated cost based on volume order? What about the cost of assembly? And should this be measured for the first item to be built, or for subsequent items if there are to be several? What about the costs of design? That is particularly tricky, since the act of designing to minimize costs itself costs money. What about cost measured in the time to produce the equipment? What about the cost of revising the design if it isn't right; this is a cost that may or may not occur. How does one assign overhead or indirect costs? And so on. In a completely particular situation one can imagine an omniscient designer knowing exactly which of these costs count and being able to put dollar figures on each to reduce them all to a common denominator. In fact, no one knows that much about the world they live in and what they care about.

The dilemma is real: there is no reducing the evaluation of performance in the world to a few simple numbers. The solution is to understand what systems objectives are: they are guides to understanding and assessing system behavior in various partial aspects. Various measures for each type of objective are developed, and each shows something useful. Since all measures are partial and approximate (even conceptually), rough and ready measures that are easy to make, display and understand are often to be preferred to more exact and complex measures. Standard measures are to be developed and used, even if not perfect. Experience with how a measure behaves on many systems is often to be preferred to a better, but unique, measure with which no experience exists.

While this book does not treat systematically all the different system measures, many of them are illustrated throughout the book. Table 4 provides a guideline, listing in one place the components that contribute to overall cost and performance.

Table 4: Cost and performance components for a system.
 (Bell, Grason, and Newell, 1972, p24)

Cost Components

Cost Components Arising From the Design Effort

- specifying
- designing (drawing, checking, verifying)
- prototyping
- packaging design
- describing (documenting)
- production system design
- standardizing

Cost Components Arising From Production

- buying (parts)
- assembling
- inspecting
- testing

Cost Components Arising From Selling and Distribution

- understanding
- configuring (i.e., user designing)
- purchasing
- applying
- operating in the environment (heat, humidity, vibration, color, power, space)
- repairing
- remodeling

Performance Components

Performance Factors Arising From Designing, Producing, & Selling

- environment
- for a single task
- for a set of tasks
- operation times
- operation rate
- memory size & utilization
- reliability, availability, maintainability, and error rate
- mean time between failures (mtbf)
- availability (percent)
- mean time to repair (mtrr)
- error rate (detected, undetected)

It is necessary to point out some conflicts among the various activities. The list of conflicts is very real and taken from experience.

System Cost Versus Component Cost. DEC sells products at each of the packaging levels of integration from chips to turnkey application systems. Because each product is constructed from lower packaged levels, and the levels model (View 3: Packaging Levels of Integration) strictly applies, it is very difficult to have designs that are optimally competitive at every level. For example, if DEC sold just hardware systems (cabinet level) it would not need a boxed version of its CPU's. The box level could then be deleted and the price of the systems product would be proportionately lower. When primitives are to be used as building blocks, there is a cost associated with providing generality. For example, some boxes are overpowered for most of their final applications because the powering was designed for the worst possible configuration of modules within the box. (Some boxes have also been underpowered as increases in logic density were accompanied by increases in power density, permitting new worst case configurations in existing boxes.)

Initial Sales Price Versus User Life Cycle Cost. There is a cost associated with parts that break and have to be repaired and maintained. Nearly every part of the computer can be improved over a range of a maximum of a factor of 10 to provide increased reliability (extended MTBF) for a price. To the extent these costs are added, the product will be less competitive in terms of a higher purchase price. However, if the total life cycle costs are considered, the product may still be better even at the higher initial cost.

Reliability, Availability, Maintainability (and Producibility) Versus Performance. By designing to take advantage of the fastest components and operating them at the limit of their capability, one is able to have increased performance. In doing so, the tradeoff is clear, producibility, reliability

(error rate), and maintainability (ease of fixing) all generally suffer.

Performance Versus Cost. This is the most traditional design tradeoff. In addition to the conventional product selection, the planning of a computer family further increases selection/tradeoff process.

Early Shipment Versus Product Life and Quality. By delivering products before they are fully engineered for manufacture, one is able to improve the ship date but at a significant risk. If faults are found that have to be retro-fitted in the factory or field, the cost far outweighs any early product availability.

Another Longer Design Versus Product Life. By taking longer to design, a product can be designed in such a way that it is easier to enhance. On the other hand, if prospective customers, especially new customers, are faced with a choice between the competitor's available non-optimum product and your non-available optimum product, they may not be willing to wait.

Operating Environment Versus Cost. Here there are numerous tradeoffs even within a conventional environment. In each of the packaging dimensions: heat, humidity, altitude, dust, EMI, etc. there are similar tradeoffs that may appeal to unique markets or may simply translate to increased reliability in a given setting. The Norden 11/34M is an example of packaging to provide a PDP-11 for the aerospace environment.

The principles of computer design and the optimization efforts associated with those principles are parts of computer science and electrical engineering,

which are the responsible disciplines for these activities. From computer science come many of the technical aspects, such as instruction-set architecture; much of the theory, such as algorithms and computational complexity; and almost all of the software design, such as operating systems and language translators, that are applied in the practice of computer engineering. However, in their construction, computers are electrical. Thus, the discipline that has fundamental responsibility is electrical engineering. Thus discussion of the Practice of Design View concludes with Table 5, a set of maxims compiled by Don Vonada, an experienced DEC engineer. Many other engineers in many other companies have developed similar sets of maxims.

Table 5 - Vonada's Engineering Maxims

1. There is no such thing as ground.
2. Digital circuits are made from analog parts.
3. Prototype designs always work.
4. Asserted timing conditions are designed first, unasserted timing conditions are found later.
5. When all but one wire in a group of wires switch, that one will switch also.
6. When all but one gate in a module switches, that one will switch also.
7. Every little pico farad has a nano henry all its own.
8. Capacitors convert voltage glitches to current glitches (conservation of energy).
9. Interconnecting wires are probably transmission lines.
10. Synchronizing circuits may take forever to make a decision.
11. Worse case tolerances never add - but when they do they're found in our best customer's machine.
12. Diagnostics are highly efficient in finding solved problems.

13. Processing systems are only partially tested since it is impractical to stimulate all possible machine states.
14. Murphy's Laws apply 95% of the time. The other 5% of the time we're on coffee break.

VIEW 7: The Blaauw Characterization of Computer Design

Another view is based on the work of Blaauw [1970]. He distinguishes architecture, implementation, and realization as three separable levels in the construction of anything, including computer structures.

The architecture of a computer system defines its functionality (behavior) as it appears to the machine-level programmer, and can be characterized by the instruction set processor, the ISP. The implementation of a computer system is the actual hardware structure - the register-transfer (RT) level behavior and data flow organization. This also includes various algorithms for controlling a machine as it interprets an architecture. Realization encompasses the actual technologies used and includes the kind of logic, how it is packaged and how it is interconnected. Realization includes all the details associated with the physical aspects of the machine.

Modern architectures (ISPs) usually have multiple (RT) implementations. For example, the LSI-11, 11/40, and 11/60 are different implementations of the same basic PDP-11 instruction set. Sometimes, although rarely, a particular implementation has more than one realization. For example, the IBM 7090 has the same architecture and implementation (i.e., the same ISP and RT structure) as the IBM 709. The difference lies in realization: the 709 used vacuum tubes, the 7090 used transistors. For a more recent example, two models of the PDP-11 architecture that share the same implementation are the 11/34 and Norden's ruggedized realization, the 11/34M. The realization differs, however, as the latter uses militarized semiconductor components, militarized component mounting, and a different packaging and cooling system.

The following table attempts to clarify the distinguishing characteristics of architecture, implementation and realization.

Table 6
Characteristics of Design Areas (Blaauw and Brooks, Computer Architecture, 1978, in preparation, Chapter 1)

	Architecture	Implementation	Realization
Purpose	function	cost and performance	buildable and maintainable
Product	principles of operation	logical design	release to manufacturing
Language	written algorithms	block diagram, expressions	lists & diagrams
Quality measure	consistency	broad scope	reliability
<hr/>			
Meanings (used herein)	ISP machine ISP	RT-level machine; Microprogrammed sequential machine	physical realization; physical implementation

(at logic level)

This book concentrates on the realization and implementation columns in the above table. Instruction set architecture is discussed only insofar as it interacts with the other two characteristics. There are also some differences between the views of Blaauw and Brooks and those expressed in this book. It is important to try to reconcile these differences, because everyone engaged in computer engineering uses the words architecture, implementation, and realization - quite often to mean different things. This book will not limit the definition of architecture to just a machine as seen by a machine language programmer. Instead, it will use architecture to mean the ISP associated with any of the machine levels described in View 2, Levels of Interpreters. Therefore, architecture standing alone will mean the machine language, the ISP. This book will also use: architecture of the microprogrammed machine as seen by a microprogrammed machine's microprogrammer; architecture of the operating system as the combined machine of operating system and machine language; and architecture of a language for each language machine. For example, ALGOL, APL, BASIC, COBOL and FORTRAN all have as separate and distinct architectures as a PDP-10 and a PDP-11 do. This use of architecture, since it describes behavior, is quite consistent with Blaauw's. Moreover, when applied to software structures, Blaauw's framework fits well. There are two implementations, FORTRAN-IV-PLUS (an optimizing compiler) and FORTRAN IV (a threaded code system), of the one ANSI FORTRAN architecture. Moreover, different implementations use different realization techniques: some use BLISS, others use assembler language.

Although Blaauw and Brooks define implementation and realization clearly,

these definitions aren't widely used. The main problem is that they are both sensitive to technology changes and hence interact closely. Computer engineers tend to overuse and intermix them so that the two words are used interchangeably. This is reflected in this book, where they are used to have roughly the same meaning (e.g., "The KI-10 was implemented using TTL H-series logic.") In the table, definitions are given for the two words to further relate descriptions back to these definitions if the reader chooses. "Implementation" is the Register Transfer level machine, roughly the microprogrammed machine, and "realization" is the physical realization, the physical implementation in terms of packaging and technology.

The most useful distinction is between architecture on the one hand and implementation (subsuming realization) on the other. Seeing the distinction clearly enables one to preserve architectural compatibility between machine models, and this is crucial if users' and manufacturers' software investments are to be preserved. Implementation can then be as dynamic as desired, being continually changed by technology. Architecture must remain static for long periods (ten years is a common goal).

Maurice Wilkes, in 1949, only one month after his EDSAC computer was operational, and before any stored program computers in the United States were operating, had already perceived the value in having a series, or set, of computers share the same instruction set. He said the following.

"When a machine was finished, and a number of subroutines were in use, the order code could not be altered without causing a good deal of trouble. There would be almost as much capital sunk in the library of subroutines as the machine itself, and builders of new machines in the future might wish to make use of the same order code as an existing

machine in order that the subroutines could be taken over
without modification."

Chapter 1 References

Popek, G. J. and Goldberg R. P., Formal Requirements for Virtualizable Third Generation Architectures, Communications of the ACM, vol. 17, no. 7, July 1974, 412-421.

Knight, K. E. Changes in Computer Performance Datamation, vol. 12, no. 9, pp. 40-54, Sept. 1966.

Turn, Rein, Computers in the 1980s, Columbia University Press, N.Y., 1974.

Sharpe, W. F., The Economics of Computers, Columbia University press, N.Y., 1969.

Phister, M., Data Processing Technology and Economics, Santa Monica Publishing Co., Santa Monica, 1976.

Asimow, Morris, Introduction to Design, Prentice Hall, 1962.

Simon, Herbert A., The Sciences of the Artificial M.I.T. Press, 1969.

Blaauw, G. A., Hardware Requirements for the Fourth Generation, in F. Gwenberger (Ed.). Fourth Generation Computers: User Requirements and Transition, Prentice-Hall, pp. 155-168, 1970.

Wilkes, M. V. A Personal Communication from M.V. Wilkes to S. H. Fuller 1/13/77 which confirmed that the quote (Chapter 1, page 58) which appeared in a British

Computer Society's History of Computing in 1949 was accurate.

Grosch 1953? (p. 32)

Bell, Grason, and Newell Ref.

Cray and Amdahl? (p.10 draft)

Bell (1971)? Def. of Mini (p.30 draft)

Baron Computing Europe 1975? (p.30)

Chapter 1 FiguresFigure

- | | |
|-----|---|
| 1 | Structural levels from Bell & Newell |
| 2 | The basic interpretive loop |
| 3 | A processing system (in text) |
| 4 | Hierarchy of interpreters |
| 5 | Pipes through interpreters |
| 6 | Levels versus time |
| *7 | Noyce's levels |
| 8 | Network, not tree |
| 9 | Three design styles |
| 10 | Evaluation from base design B |
| 11 | Price versus time for lack machine class |
| 12 | (Groschlin) |
| 13 | Histogram of data-type by application |
| *14 | Philosophy of design |
| *15 | Product development schedule I (Figure 4.21.3) |
| *16 | Product development schedule II (Figure 4.21.4) |
| *17 | Engineers assigned versus elapsed time |

Tables

- | | |
|---|---|
| 1 | Four levels |
| 2 | Using new technology for constant cost and constant performance designs |
| 3 | Discipline/environment based functional segmentation scheme |
| 4 | (BGN) cost/performance components for a system |
| 5 | Vonada's engineering maxims |
| 6 | Characteristics of design areas (in text) |

* Lifted; supply credit line