A USER LOOKS AT DA -- YESTERDAY, TODAY, TOMORROW

A. E. Fitch
International Business Machines Corporation
Systems Development Division, Poughkeepsie, New York

Summary

While much has been written about the technical detail of Design Automation, little has been written about how a "user" views DA.

This paper will not address the technical details of Design Automation, nor will it consider the entire range of DA facilities and users in existence today. Instead, the paper will contrast a specific user's needs to the facilities available to that user yesterday and today. This contrast will follow a general definition of the term DA, and a detailed description of the various tasks confronting the particular user. For the purpose of this paper, the user's job is to develop a computing system.

The question of automation versus assistance is discussed, especially in light of this paper's projections for what DA must supply to the user of tomorrow. For purposes of this paper, yesterday is the time of discrete wiring, today involves printed wiring, and tomorrow is the era of integrated circuits.

The importance of the tools, which tomorrow's DA must supply, is put in perspective by a discussion of a factor called "the pressure point." A general characterization of the DA authors, DA users, and procedures used in the three time frames of this paper is made.

The emergence of a mixed breed of programmers and engineers is noted. The importance of this mixed breed to tomorrow's DA is discussed.

Although this paper deals with a very specific kind of DA, the observations and projections undoubtedly have fairly universal application.

Introduction

Those of us who have grown up with DA have come to consider it a fact of life. Regardless of whether DA means Design Automation, or Design Assistance, we understand the why and what of DA. The fact that this familiarity does not exist universally was brought home to me recently.

After I had completed a presentation dealing with DA to a group of engineers, a young, newly-hired engineer came up to me and said, "But there's one point I missed. What is DA?" In order to establish a common understanding of what DA means in this paper, I define DA pictorially as shown in Figure 1. DA is a tool used by the engineer:

1. to record the design.
2. to check some of the design.
3. to perform certain of the well-defined design chores.
4. to provide management with information.
5. to provide manufacturing organizations with the information required to produce the product.

Although this definition of DA implies a fairly sophisticated system, I believe that most, if not all DA activities tend to fit somewhere in this general definition, regardless of whether the object for action by a given system is a computer or a circuit, a bridge or a bikini.

I will not deal with the technical detail of DA. Nor will I describe or even hint at any technological breakthrough. Instead of making such revelations, I would like to spend a few minutes looking at where we were, where we are, and where we should be headed. Because DA is useless unless a "user" exists, I will first establish a user. For purposes of this presentation, the "user" is actually a composite of many individual users whose collective job is to develop and produce a computing system. The tasks that this user generally performs will be contrasted with the general capabilities of DA of yesterday and today.

I will consider which of these user tasks needs attention in tomorrow's DA. While I realize that the general term DA encompasses a universe much wider than that which I will address, I suspect that some of the observations, philosophies, and projections I will make can find application in the areas of DA not directly addressed here.

How the User Views DA

Before we examine the user's job in detail, I think it would be interesting to see what mental image four different groups of people might have of DA. The mental image that comes to the mind of the new engineer is most likely that which is shown in Figure 2. This potential user does not

know which end of the monster to grab first and is afraid of being overwhelmed by the whole thing. I am sure this is the image that came to the mind of the young engineer I previously described. The project manager who has to fund either the creation of or operation of a DA system might have a view as shown in Figure 3. While DA is of great importance, it isn't necessarily cheap to create or inexpensive to use, it costs money to make tools, and the lack of proper tools can cost time, which is also expensive.

The programmer who created the DA tool probably sees the tool as shown in Figure 4. It is perfection personified and can do no wrong. The experienced programmer, however, knows that jewels can have flaws. The experienced user will view DA as all of the above at one time or another but upon sober reflection will probably view DA as shown in Figure 5. There is no question but that the user, given adequate resources, can do everything that DA tools can do -- maybe better -- but that he recognizes the value of the tool and the impracticality of doing without it. He may even come to appreciate the value of a few of the ways in which the tools force him into what seems to be regimented thinking, although he probably would never admit this -- especially to the person who created the tools.

Facilities Available to User

Yesterday

For the purposes of this presentation, I consider yesterday as 1958-1962. The predominant technologies involved mostly unit logic elements (generally cards) on which discrete components were mounted. These cards were inserted in panel sockets, with intrasocket and intersocket connections made with discrete wiring. In this time frame, few functional logic cards existed and the printed circuit patterns required for use on cards were derived from manually generated artwork. Installation of yesterday's engineering changes (EC's) meant a slightly different use of the normal manufacturing tools, and presented no insurmountable technical problems. Speed was generally measured in microseconds and technology oriented rules were few in number. Signal delays due to wiring were not a major concern.

Today

In the context of this presentation, today is defined as 1962 to the present and is characterized by a continuation of the use of unit logic cards and the introduction of a larger number of functional cards. Both of these card categories employ hybrid components in place of many of the discrete components of yesterday. The sockets and panels of yesterday have generally been replaced by multilayer printed circuit boards. The discrete wires have for the most part been replaced by printed circuit wiring. Engineering changes today are possible but inconvenient because they require special tools and procedures owing to the nature of the personality to be changed. Speed today is sometimes still measured in microseconds but more generally it is measured in tens or hundreds of nanoseconds. The number of technology-oriented rules to be adhered to has increased markedly. Wire lengths measured in feet can be cause for concern.

Tomorrow

Tomorrow is probably upon us. It is characterized by the higher component densities we will experience in SSI (Small-Scale Integration), MSI (Medium-Scale Integration), LSI (Large-Scale Integration), and the ambiguous Utopia all engineers seek, RSI -- just the right scale of integration. Tomorrow's engineering changes, will require tooling of extreme complexity and indeed may mean remanufacturing rather than reworking. Tomorrow's speed will be measured in nanoseconds and picoseconds. Inches of wire will cause concern, and the volume of technology-oriented rules is liable to become unmanageable.

Tasks Confronting the User

In performing his total job, the user must perform many tasks, some of which are not now addressed by DA and maybe never should be. But to fully describe our composite friend, I will briefly discuss all elements of his job. The degree to which a given user will address each element is a function of many things including the tools available to help him, the complexity of the product, and scheduling considerations. His activities are listed in Figure 6 and are described as follows:

1. Plan and specify the product -- In this phase the major input considerations will be the job to be done by the product, and a target for the product's final cost. This step will determine the choice of the technology to be used and a plan for implementation.

2. Develop and Record second-level product descriptions -- This step will lead to the selection of algorithms, the definitions of functional elements (e.g., I-Box, E-Box, etc.), and internal machine strategies required to meet cost and performance objectives. The degree of descriptions produced in this step may vary from a few

second-level drawings to an algorithmic description of the entire product.

3. Verify the second-level or algorithmic operation -- Execution of this step may be as simple as a review of second-level diagrams, or as complex as a total algorithmic simulation of the product. This is the first of many places where the question "Is it right?" is asked.

The preceding steps are generally referred to as "architecture." The following steps are generally referred to as "implementation." These steps can be compared to the reduction of an artist's rendering of a building, to the detailed plans necessary for its construction, to the dimensional checking of such plans, and to the testing of the plans for structural strength. Although these steps are shown as distinct operations, it should be understood that there are varying degrees of interaction between them.

4. Develop and record the detailed product logic -- This design step sees the engineer reducing the second-level or algorithmic description of the product to detailed logic drawings or logic lists, considering such factors as technology rules, permissible logic chain length, and interaction with other areas of design.

5. Check the detailed product logic -- In this step, the engineer tests the detailed logic against specific examples to prove that the algorithm has been reduced properly to detailed logic. (Mistakes, of course, are always in the recording of his work, not in the work itself.) This test is generally a DC functional check, and is sometimes referred to as desk debugging. Again we ask: "Is it right?"

6. Package the detailed product logic -- At this point, the engineer maps the logic into specific physical entities. This step may involve choosing prepackaged logic elements (e.g., cards) and "placing" these on a panel (e.g., board), or it may involve the segmentation of the detailed logic into specific functional packages (e.g., cards) that will be unique to this product. The packaging information is recorded on the detailed product logic documents at this time. The considerations in this step may range from maximum performance to minimum cost and are highly product- and technology-oriented.

7. Check recorded package detail -- The probability for error in the packaging activity is relatively high; not on the part of the engineer, of course, but in the recording of his decisions. Owing to that fact, and the fact that we have not yet found the way to make two physical elements occupy the same space at the same time, this step is executed. It also helps to avoid embarrasment and some engineering changes at a later time.

8. Check the dynamic characteristics of the packaged logic -- One of the major considerations used in developing the detailed logic was the minimum and maximum length of logic chain that could be tolerated. This requirement also served as a consideration in packaging the logic. At this point in his activities, the user tries to prove that the absolute delays through his logic chains are within the established bounds. The degree of checking may range from analysis of a few networks he suspects are critical to a complete analysis of all the logic. Once more we ask: "Is it right?"

9. Provide the information required to manufacture the product -- Now the recorded design information contains the full product description; bills of material and detailed manufacturing instructions can be generated. The product can be built. At this point in time, our friend deserves a vacation. He has been very busy, regardless of DA.

10. Repeat all or part of the above for EC's -- Our friend's vacation is soon over. The errors he didn't find previously are obvious now that hardware is standing. How could he have let that obvious error get by his eagle-eye? This phase involves correcting the design, generating rework instructions to modify the existing hardware, and thinking each change is the last. On the same day he corrects the last error and is worried about what to do next, someone is sure to request that an additional feature be designed -- so our friend is back in business again.

DA Applications

Yesterday

Given the total job the user must do, let us now examine the areas in which DA has been, is, and must be used. Figure 7 contrasts the user's total job with the DA capabilities he had available yesterday. Generally speaking, DA provided the ability to record the design, including packaging data, check the packaging data, and produce information for manufacturing the product. In effect, yesterday's DA replaced the many clerks and draftsmen who would be required to manually document the design, and DA provided a bonus of waving a red flag when the recorded design violated basic ground rules. While simulation was being discussed yesterday, DA users and creators were too busy cutting their teeth to have time to pursue it. The relatively low speed of the circuits used, plus the fact that EC's seemed to be a

practical way of life, put AC performance testing far from the minds of most people. The DA of yesterday gave the user some degree of support in 50% of his activities and represented a major step forward despite this hindsight view of its capabilities.

Today

In Figure 8, we see the capabilities of today's DA added to the chart. Significant improvements have occurred in capabilities that existed yesterday. The ability to do some design checking has been introduced. Not only has the ability to record detailed packaging information been improved, but first generation programs now exist that try to relieve the user of some of the packaging job. Because these programs are first generation, the thinking has not been deep enough to cover the entire range of users. But progress has been made. More complicated packages require more comprehensive physical checking programs, so DA capability in this area has also improved. For the first time, a capability exists to examine the product in a dynamic sense. Manufacturing data is more complex today, and DA has provided improved tools to create and handle this more complex data. DA has learned, and continues to learn, how big a problem the processing of EC's can be. Today's rework instructions are more important than ever because the number of influencing factors is greater today than yesterday. Today's DA supports the user to some extent in 70% of his activities including all of those associated with detailed design.

Tomorrow

In Figure 9, we see the initial capabilities that will be required of tomorrow's DA added to the chart.

In considering what tomorrow's DA must do, it is urgent that we consider the two meanings of DA. Undoubtedly there are those who view tomorrow's DA as being the ultimate in automating all the activities with which the user is faced. To these people, DA means Design Automation. I propose that the "A" in DA be capitalized for Assistance and lower cased for automation. There are tasks such as packaging that may be distastful to the user, but that he can do with reasonable effort. There are tasks such as dynamic performance analysis that are nearly impossible for the user to do without having the proper tools.

As I look toward tomorrow, my fundamental belief is that the designer should be left to do the creative thinking that a job like partitioning requires, and that the energies of DA organizations should be aimed at providing maximum assistance in his handling of job details.

The areas of planning, algorithmic design, and algorithmic checking deserve attention but with a small letter a. These activities tend to be highly tailored to the particular product. While some projects may find it efficient to use general-purpose programs, larger projects will actually save money by creating their own tools. Tools to assist in logic design and packaging must improve, but the tools available to check the design (DC, packaging, and AC) must improve to the greatest extent. These tools must receive attention with a capital A.

It may seem surprising that I have not projected the need for fully automatic packaging capabilities or the absolute elimination of a user manipulating anything but a higher-level design language. The fact is, however, that design verification is today, and will continue to be tomorrow, the single most important interactive aspect of DA. Perhaps in the future, manufacturing instructions will automatically be created on the basis of an algorithmic description of the product, but it seems hard to believe that such a Utopia can be justified economically in light of rapidly changing technologies and requirements.

Tomorrow's technologies provide an interesting study in contrasts regarding DA support for EC's. An integrated circuit element itself requires little DA support to generate rework instructions since it is doubtful that it will be reworkable. However, levels of packages that can be reworked will require comprehensive support with the input to such tools taking many forms ranging from a change to the documented logic, to specific instructions regarding the characteristics of changes to a given physical network.

The Pressure Point

To demonstrate the importance of checking tools, I would like to describe what I will call the "pressure point." I am sure a pressure point exists in all projects, whether the output is software or hardware. In engineering projects, the pressure point is that point in the design sequence where project management exerts enough pressure on the user to force the release of a design for manufacturing, or for integration into a larger design area. Figure 10 shows a simple representation of the pressure point and lists factors that influence its position. Factors that can move the pressure point earlier are project schedules, a technology in which EC's can be

made with ease, the ease with which an engineer gives in to management pressure, and -- significantly -- the lack of tools available to the engineer to prove his design. The pressure point moves later in time as a function of difficulty in installing EC's on the hardware, the ability of the engineer to resist the pressure, and the availability of and quality of tools with which he can prove his design.

Figure 11 shows where the pressure point was yesterday, where it is today, and where it must be tomorrow. In yesterday's time frame, EC's were assumed to be a way of life because the lack of tools to prove the design left hardware test as the only proof of design correctness. Today, EC's are possible but inconvenient and they are becoming more expensive. Today we have some basic tools with which to prove the design; we are starting to place more emphasis on the ship date and less on the power-on date. Today the pressure point is moving out in time. The fact that basic tools exist is but one factor. To bring about this reorientation, we must establish the user's confidence in these tools. They must be usable. The finest screwdriver bit is useless if it is mounted in a handle that is full of burrs, just as the finest program is useless if the user must turn cartwheels to use it.

Product designs using tomorrow's technologies must have an accuracy greater than anything known before. A mistake cast in hardware will certainly require more than a soldering iron and a razor blade for correction. If adequate tools are not available, or if such tools are not properly interfaced to the user, it may be necessary to build tomorrow's product in four steps:

1. Build a discrete component model of the product, using today's technology to detect and eliminate basic design errors (e.g., Boolean errors).
2. Build the product in the final technology to detect and eliminate dynamic errors masked by the technology used in step 1.
3. Build the final product.
4. Pray very hard when designing special features after the product is built.

If we have to do business this way, DA will have failed in its assistance role and will be reduced to a record keeping system. To prevent this kind of regression, tomorrow's DA must go so far as necessary to allow execution of programs (which eventually will be run on the constructed hardware) on a complete model of the logic. This model should take into account the AC characteristics of the packaged design and the technology where necessary. This sounds like a tall order, and indeed it is. I am not oblivious to the peripheral problems of this kind of undertaking, but in my estimation it represents the major step forward that must be taken and should be the key goal for the future.

Mere lines of code assembled into programs are not enough to provide this tool. This must be an engineered tool, polished to perfection. The execution time for using this tool must not be such that it is prohibitively expensive to use. Regardless of what the object for action is in the DA activity with which each of you are associated, tools for checking the design must come first, tools for automating the design last. If you do your job well enough, tomorrow's tomorrow may be the day that hardware changes can be eliminated.

A New Breed

Perhaps you haven't recognized it, but people and procedures have changed within the time frames described in this presentation.

Yesterday saw DA tools that were created by a few programmers and used by a few engineers through a group of operating specialists. These specialists buffered the engineer from the complications of DA. The path was relatively long, and the engineer knew only the most fundamental facts of the why and what of DA. Generally speaking, yesterday's engineers only engineered. They depended on DA programmers to supply tools in the form of programs.

Today the long path is only a little shorter, but new paths have been introduced. The engineer today, in using checking tools, is much closer to DA than he ever had been previously. A new breed of user is developing. This breed is complemented by a new breed of DA programmer. Although there are still many engineers and programmers around, today we see the emergence of this new breed that I will call "engigrammers" and "progineers." No longer are some engineers at best novices at programming, nor are all programmers novices at engineering problems. Today this mixed breed is involved in understanding and solving common problems.

Today the engineer is no longer totally dependent on programmers to supply the tools he needs. The universal acceptance of higher-level languages, coupled with the general recognition on the part of the engineers that it does not take black magic to create specialized tools for limited applications, has enabled the engigrammer to create his own tools in cases where DA

couldn't, wouldn't, or shouldn't deliver the tool.

Tomorrow's user must have DA at his fingertips. The long communication paths must be nonexistent.

Tomorrow the ranks of those who create the tools and those who use the tools must be made up of a few pure programmers, a few pure engineers and a predominance of the mixed breed -- engigrammers and progineers -- with the engigrammers providing unique one-shot tools and the progineers providing the comprehensive, general-purpose tools. Both will be working from the common product file in a majority of cases.

Conclusions

In summary, yesterday's DA opened the door. It provided a set of basic tools to the user that put major emphasis on assistance, and minor emphasis on automation. Yesterday it was "da" in lower case letters.

Today's DA is yesterday's child in adolescence. In many cases, attempts to automate have received more emphasis than those to assist. Today it is "Da" with a capital D. Tomorrow must see this adolescent grow and mature. Tomorrow it must be "DA," all upper case. The lines of communication must be kept short and the user involvement in definition -- deep. Tomorrow's DA must balance assistance and automation. It must provide Assistance in the areas of extreme need, Automate the well-defined processes on a user need basis, not on the basis of arbitrary choice.

As we look to tomorrow, I would propose that two factors be well balanced. These factors are Evolution and Revolution. The creation of Revolution where Evolution was needed or the settling for Evolution where Revolution was needed, has to be avoided. As we move toward tomorrow, a reasonable balance between the blue-sky activities and the bread-and-butter jobs must exist, because a man who doesn't have the proper basic foods of bread and butter can hardly appreciate the blue sky.

In a well-known work, the idea is presented that just to keep up, one must run very fast, and to get ahead means to run even faster. This thought is certainly appropriate for tomorrow's DA.

As we look to tomorrow, we must think "big" -- we must think of tens of thousands of circuits instead of hundreds of circuits.

We must think "small" -- think of these circuits being packed into a tiny area, and what effect this will have on the DA user. We must think of the cost of a mistake.

We must think "fast" -- provide tools that allow the user ready access, provide tools that operate as fast as possible. Unless we think fast, thinking big may be a waste of time.

We must think "slow" -- be slow to jump to the conclusion that our wonderful idea is the answer to the user's prayer especially if we don't know what he is praying for. We must think of the slowness with which the tools are created and be sure that the fact that things do change with time won't eliminate the need for the tool before it is operational.

Tomorrow's DA must provide tools that are complete and comprehensive but not complicated. These tools for tomorrow must be created by a few programmers and many progineers to enable the few engineers and many engigrammers to move the pressure point to the proper point.

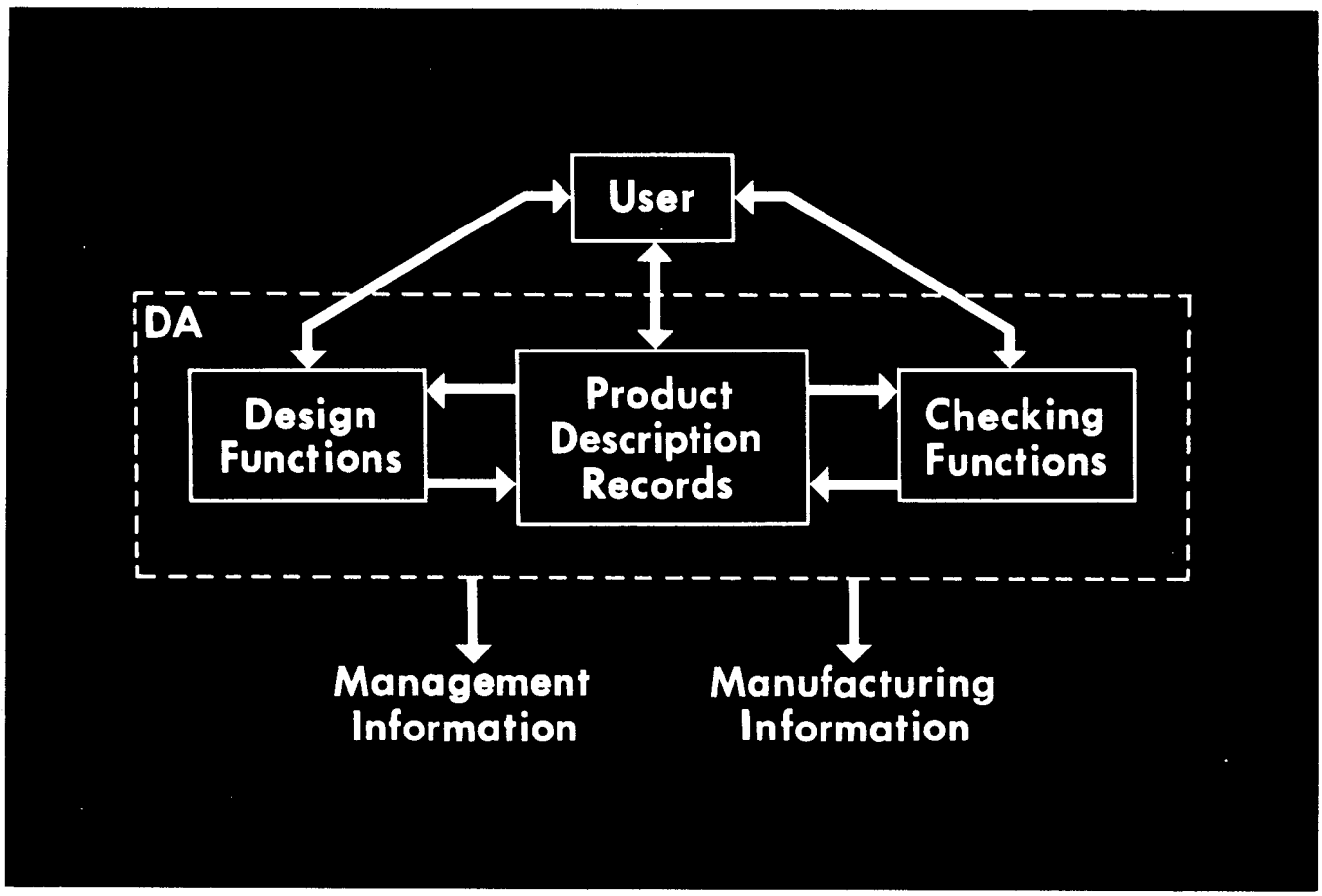

Figure 1.

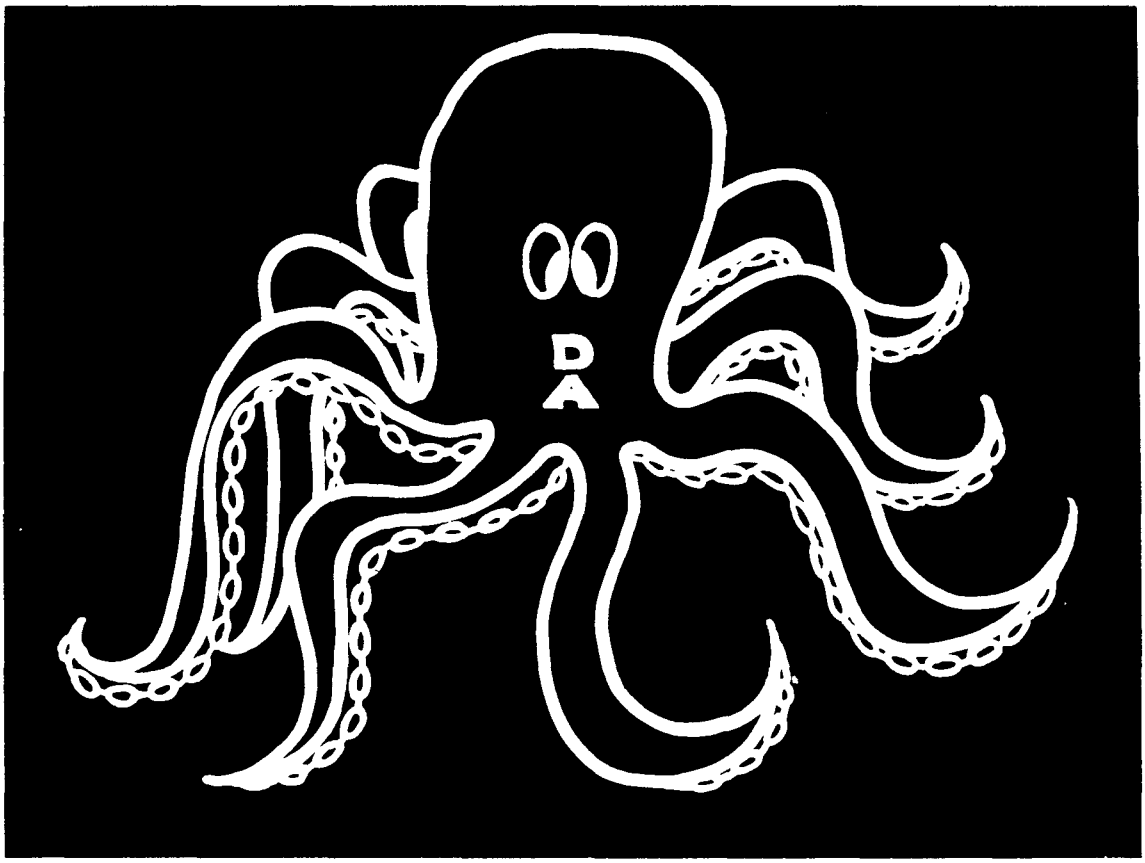

Figure 2.

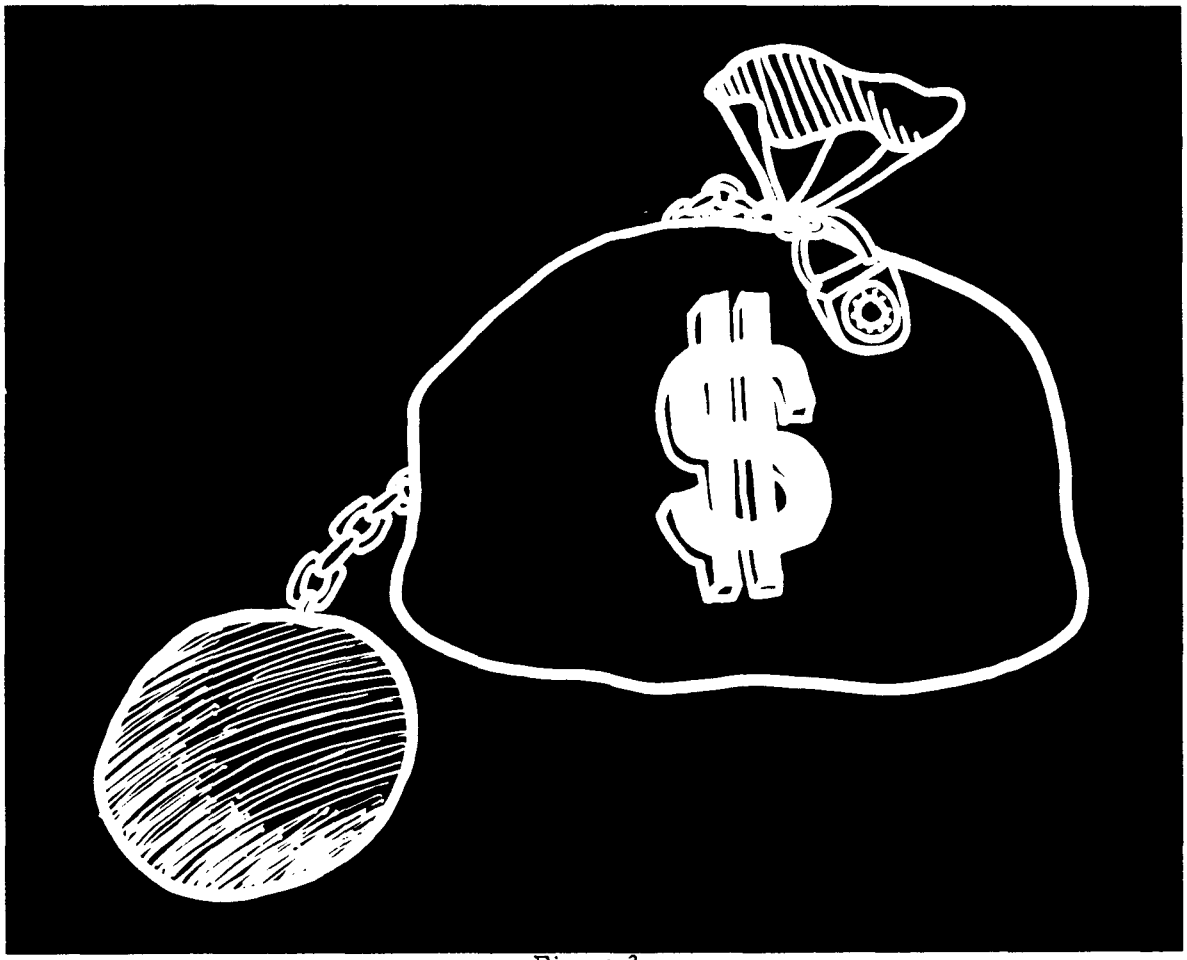

Figure 3.

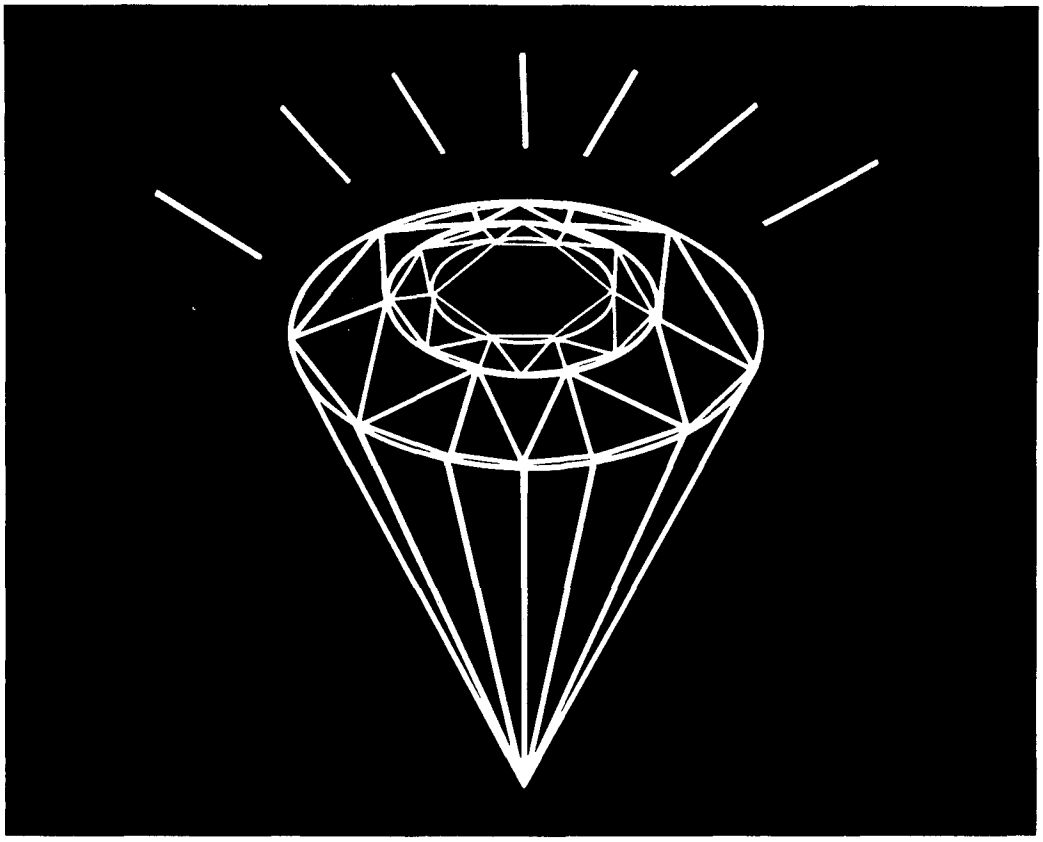

Figure 4.

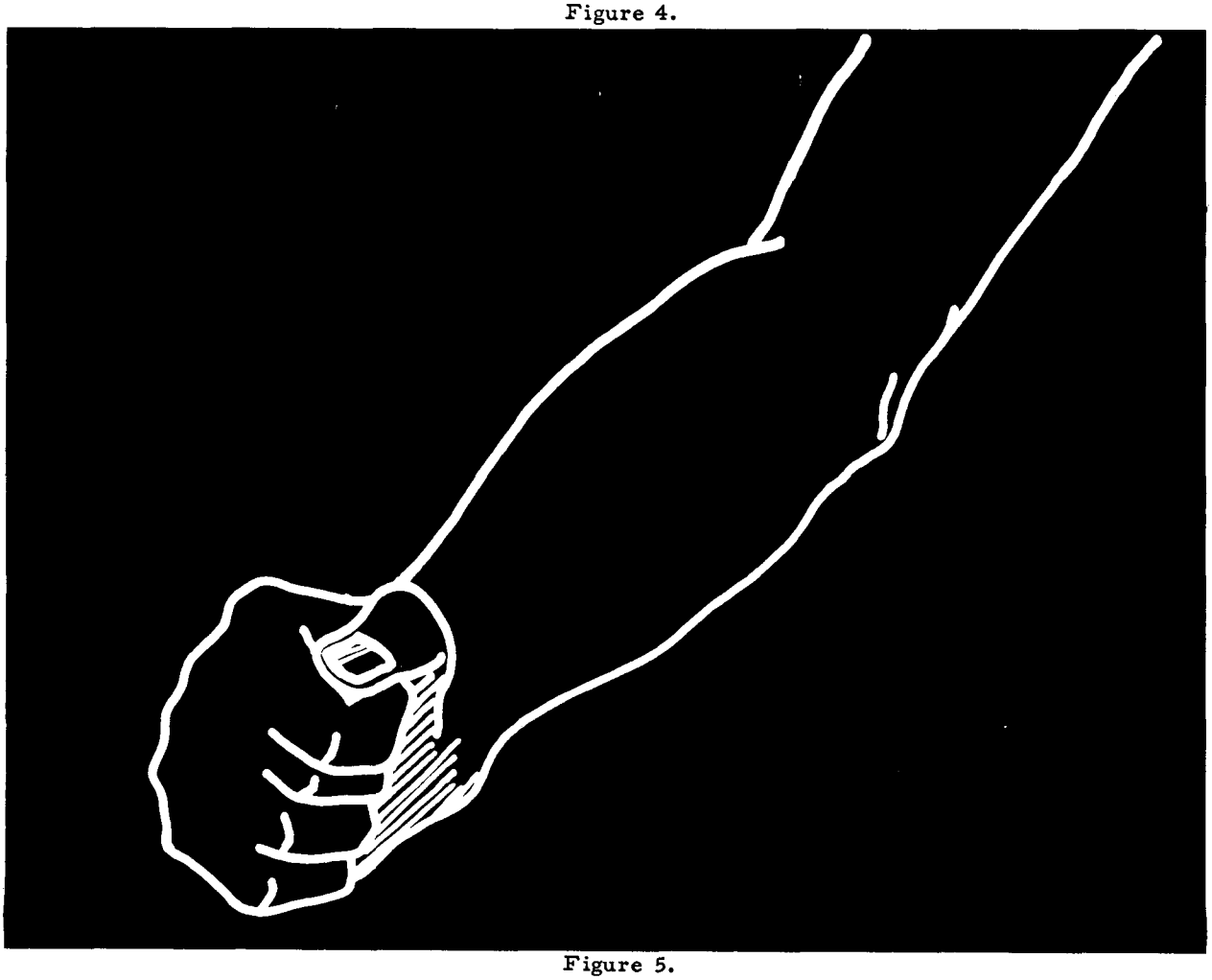

Figure 5.

1. Plan and Specify the Product.
2. Develop and Record Second-Level Descriptions of the Product.
3. Check the Second-Level Descriptions.
4. Develop and Record Detailed Logic for the Product.
5. Check the Detailed Logic.
6. Package the Detailed Logic.
7. Check the Packaging Information.
8. Check the Dynamic Characteristics of the Packaged Logic.
9. Provide Information for Manufacturing the Product.
10. Process Engineering Changes for Corrections or New Features.

Figure 6.

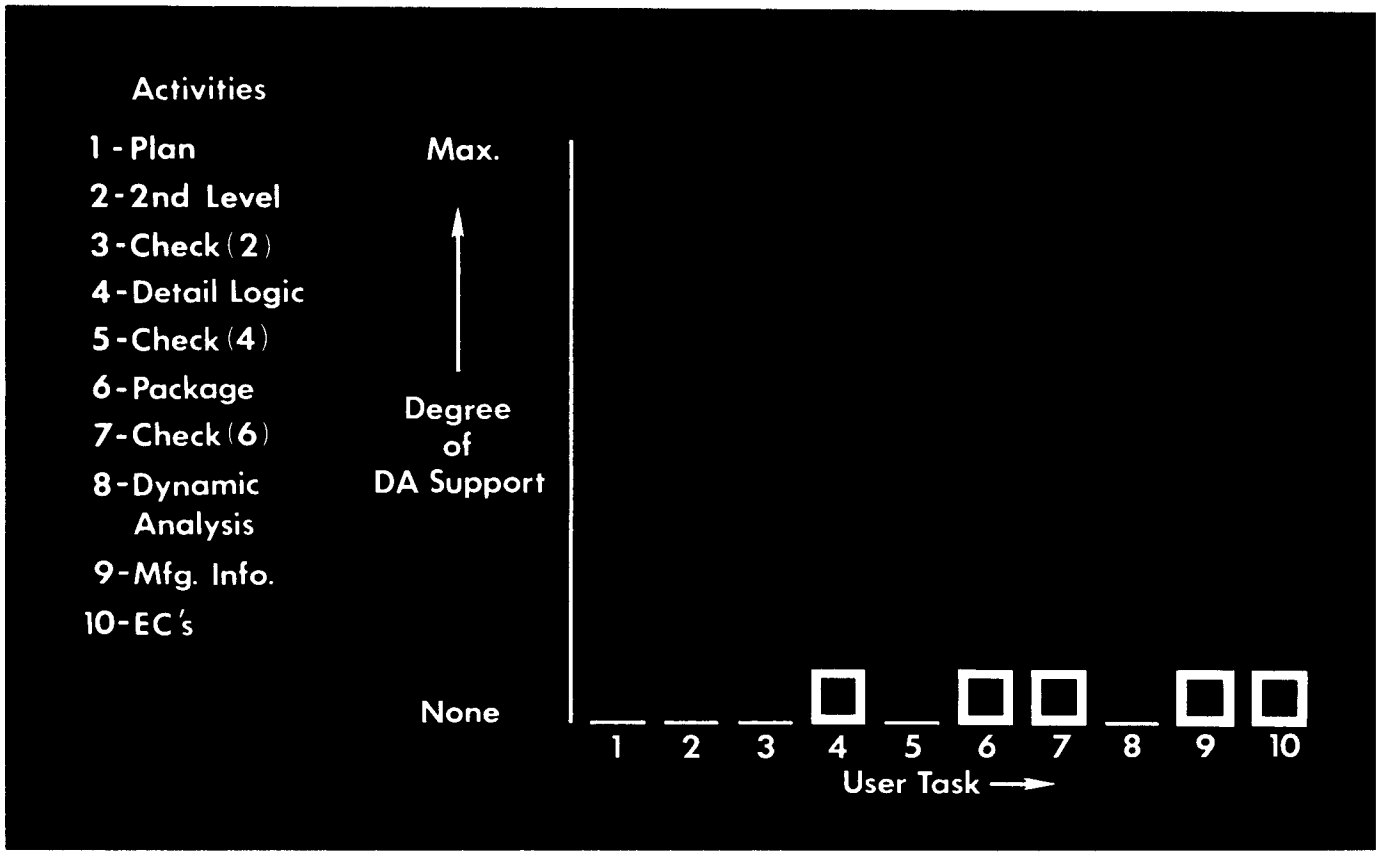

Figure 7.

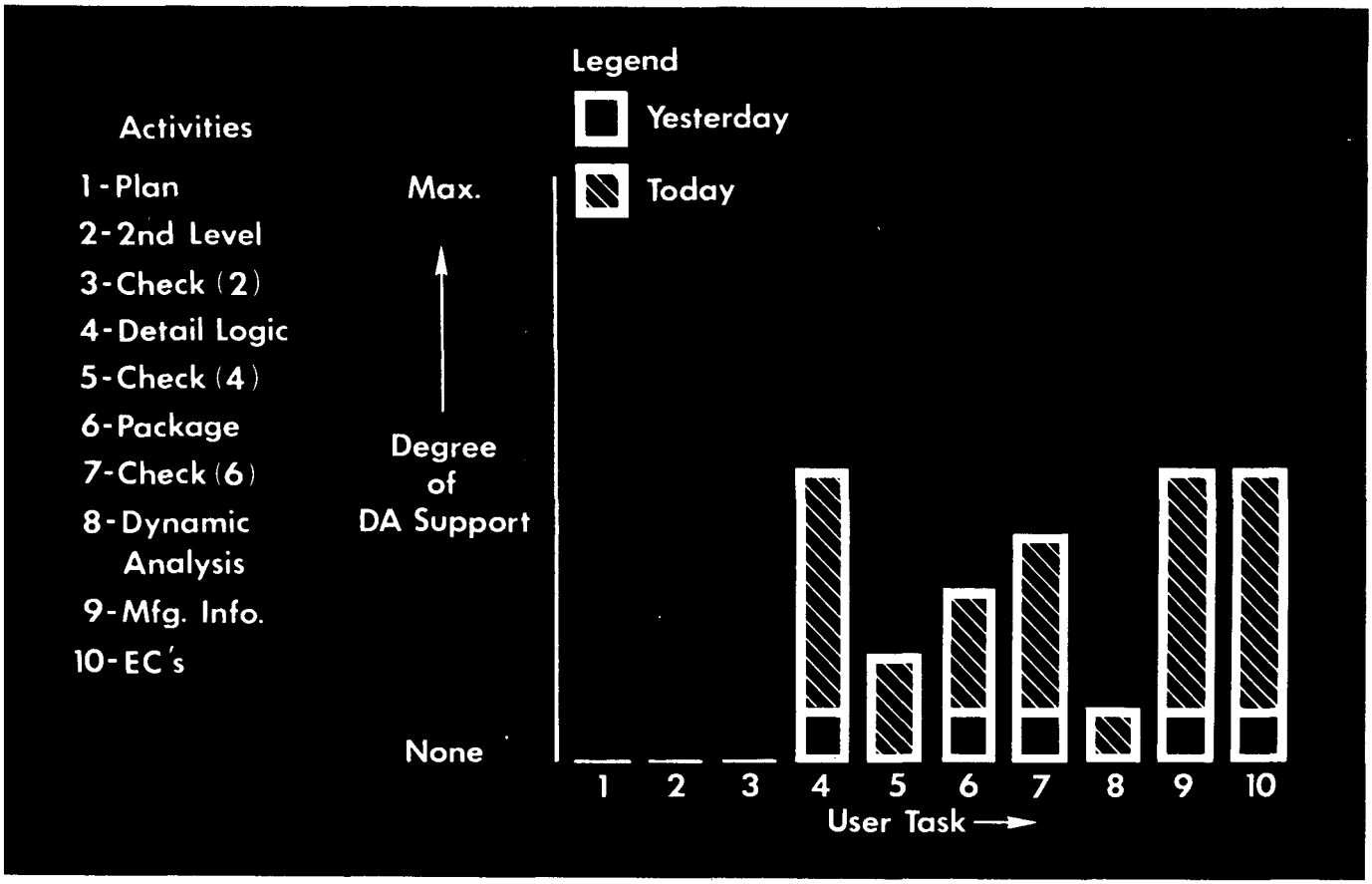

Figure 8.

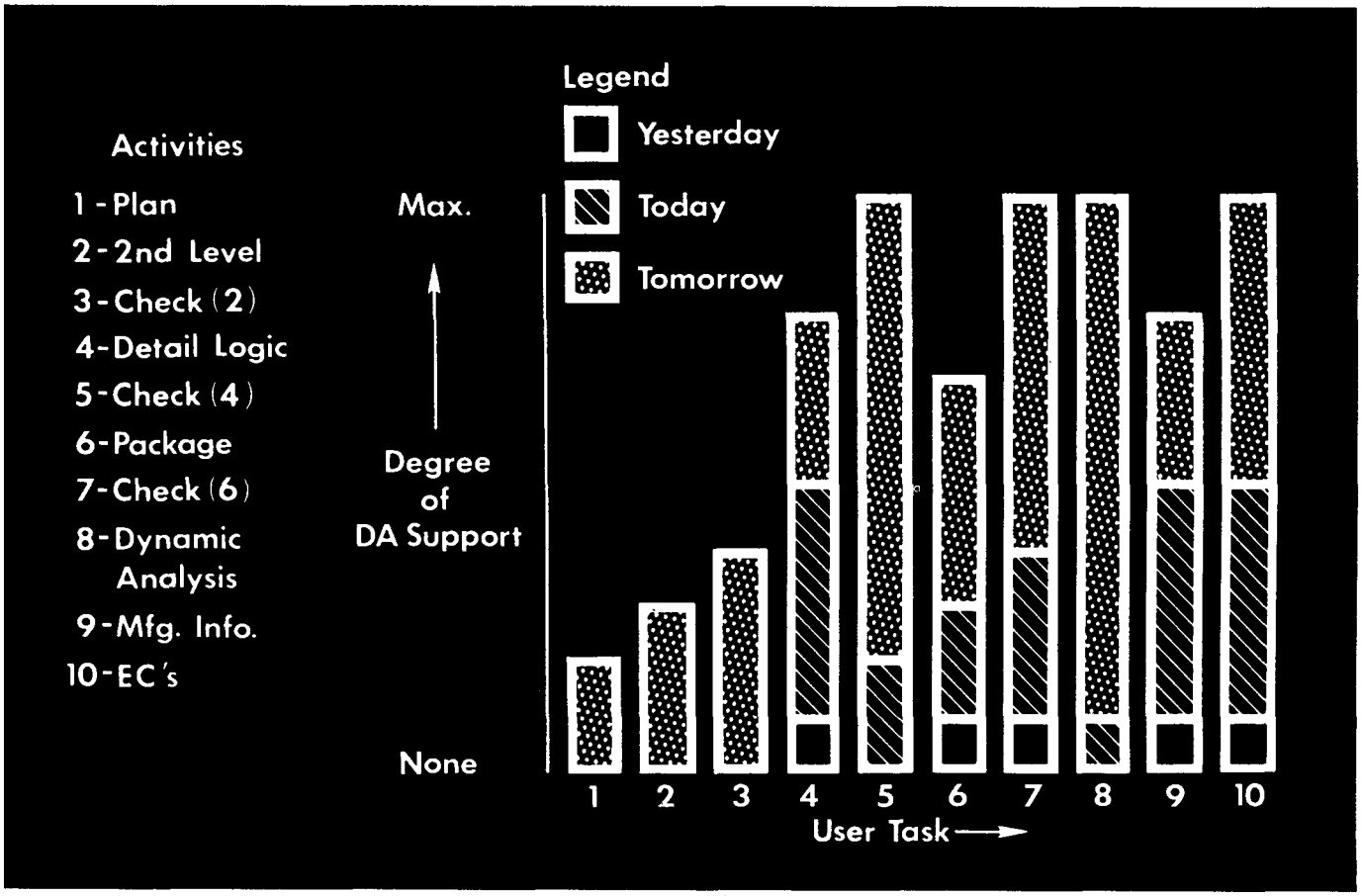

Figure 9.

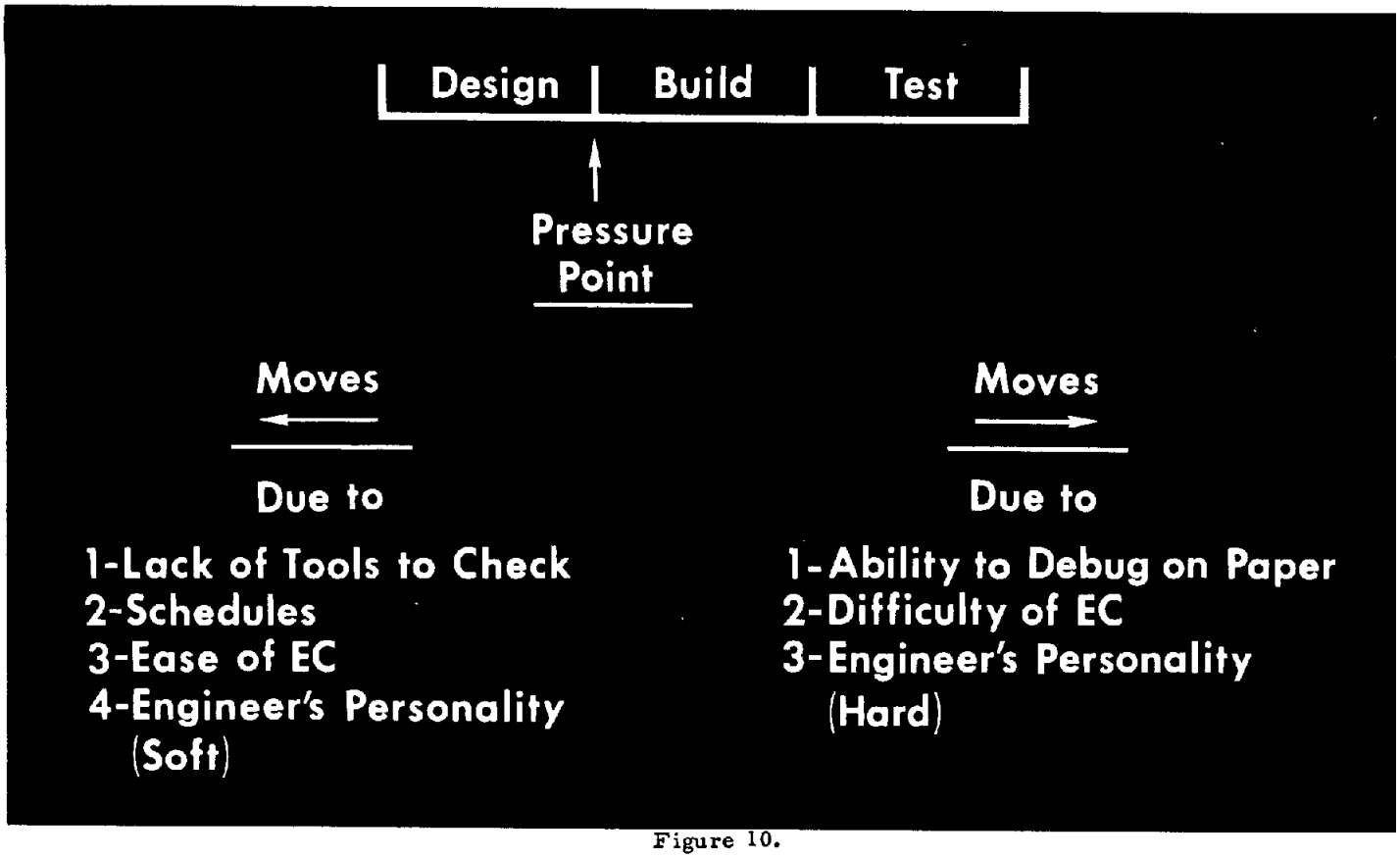

Figure 10.

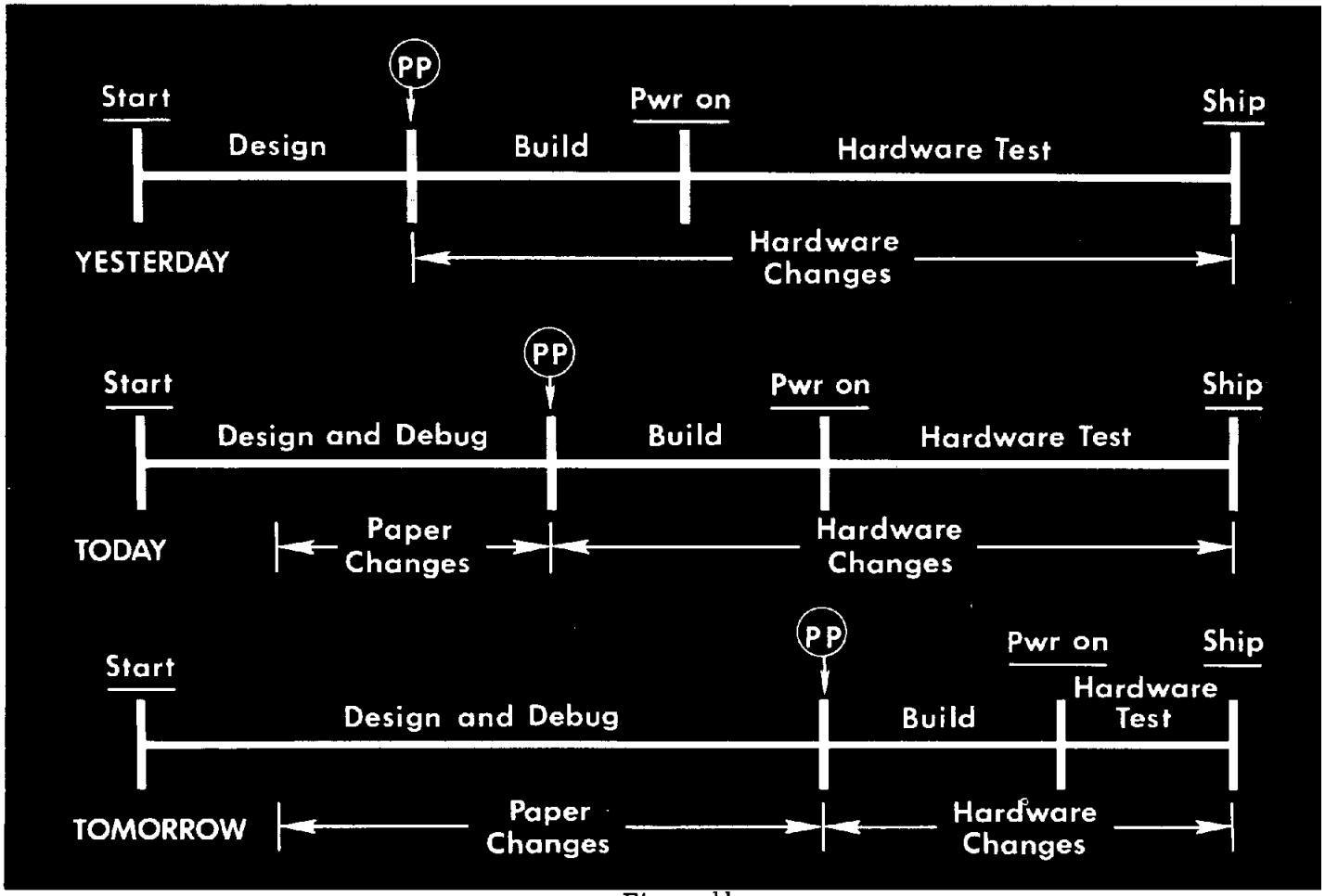

Figure 11.