EDP ANALYZER
© 1979 by Canning Publications, Inc.

JANUARY, 1979
VOL. 17, NO. 1

THE ANALYSIS OF USER NEEDS

We continue the discussion of our ideas on what the computer environment will be like in the early 1980s. As we mentioned two months ago, we expect that automated methods to aid system development will be key features of a new generation of computers. And note: these automated methods should become available on mini and micro computers, not just on maxis. And what will these automated methods be like? To illustrate, we will consider two powerful methodologies in this report that we think represent the direction that automated system development methods will take. Both aid analysts in making more complete, accurate definitions of user needs. While not yet automated, both can be—and, we expect, will be. Now is a good time to become familiar with them.

In this report, we will discuss the characteristics of two new methodologies—IA (Information Analysis) and the SA portion of SADT (Structured Analysis and Design Technique)—that seek to aid analysts in developing better definitions of user needs. In addition, we will briefly describe some of their user experiences.

To set the stage for this discussion, though, we will briefly review IBM's HIPO (Hierarchy plus Input, Process, Output). As with most IBM products, HIPO is quite widely known. Because it is widely known, it provides a basis for comparison. Both IA and SA have points of similarity and points of difference with HIPO; it will be helpful, we think, to point out those similarities and differences.

The hierarchy part of HIPO is provided by the hierarchical chart of functions of the system being studied, developed by a 'functional decomposition' of the system. Each box on the chart is named with a verb-noun combination, such as 'compute net pay.' The functions are sub-divided into lower level boxes; thus 'compute net pay' might be sub-divided into 'compute gross pay' and 'compute deductions.' By the numbering scheme used, the chart can also be used as a 'visual table of contents' for the IPO charts that follow.

For each box on the hierarchy chart, an IPO (input, process, output) diagram is developed. An IPO diagram has three rectangles drawn on it. In the left rectangle are listed the various inputs to the process. In the center rectangle, the process is described in almost pseudo-code style. In the right rectangle, the outputs from the process are listed.

The lowest level IPO charts tend to have the straight forward, detailed procedures, such as how the gross pay for regular hours worked is computed. The higher level charts tend to have flow of control procedures, such as PERFORM, CALL and IF statements.

The only attention that is paid to data is through the naming of the records related to the input and output of a process.

Jones (Reference 1) points out that there can be several levels of IPO charts—requirements IPOs, design IPOs, programming IPOs, and documentation IPOs. By implication, the requirements *hierarchy* is carried over, in which case the general structure of the solution would be the same as the structure of the requirements functional breakdown.

Benefits. Users of HIPO have noted several benefits that they have obtained. It encourages a functional breakdown of the problem area, in a top-down fashion. The charts are relatively simple to comprehend and (if little or no jargon is used) can provide good communications with users. And when combined with structured walk-throughs (which we discussed in our November 1977 report), they tend to assure users that all requirements have been defined.

Complaints. But HIPO has been the target of a good number of complaints. For one thing, HIPO is primarily a documentation technique, it is claimed, with no prescribed method of use. It is up to the user of HIPO to decide *how* to get user requirements. There is no 'configuration management' for the different generations of IPO diagrams, as they are corrected and modified. The IPO charts do not lead naturally to code; they are weak as logic tools, so they often must be supplemented with flow charts. Top-down data analysis of data is not provided, to go along with the functional analysis. It is annoying to have to write the process logic on each IPO chart. The charts tend to be large and cumbersome, and it is hard to trace the flow of data through a set of charts. So say the critics.

HIPO is a *disciplined* approach to functional analysis, and most disciplines seem to draw the wrath of analysts and programmers. So perhaps some of the above complaints can be traced to this natural resistance to discipline.

But it is also possible that HIPO has some inherent weaknesses which make it vulnerable to such attacks. Later in this report, we will give our ideas of why HIPO has been so criticized.

Before discussing SADT and IA, we should make a point about them. Both are aimed at helping the analyst *create* the definition of user needs, not just documenting the needs once they have been defined. Both methods seek to support the analyst's mental processes. Now let us see how SADT does this.

SA—Structured Analysis

SADT (Structured Analysis and Design Technique) is a proprietary methodology that has been developed by (and the acronym trademarked by) SofTech, Inc., of Waltham, Mass. Work on the methodology began in the 1973-74 time period, and has been continuing since that time.

We attended a seminar on SADT, presented by SofTech at the 1978 National Computer Conference, and we have studied some of the SofTech material on the methodology. In addition, we have drawn upon papers by Ross (Reference 2), Combelic (Reference 3) and Combelic's presentation at the 1978 National Computer Conference.

Although we have referred to SADT, it is only the SA portion—structured analysis—with which we will be concerned in this report.

The methodology includes (a) a graphical language for building models, (b) a method for developing those models, and (c) management practices for controlling the development of the models. The idea here is that the analyst (the 'author,' in SADT terminology) gains a deeper understanding of the system by developing these graphical models than is true of conventional system study methods.

There are several characteristics of the methodology to note. For one thing, it seeks a top-down decomposition of the problem area by way of modelling the area in the graphical language. (True, it is almost impossible to do strictly top-down decompostion; some bottom-up work almost always occurs—but the top-down philosophy prevails.) Further, a number of different models may be used in the development of a system—functional models to define *what* the system must do, implementation models to tell how, conversion models, and so on. Also, SA deals with the dual aspects of activities and data. Both are modelled top-down, using the same graphic language.

The inspection of intermediate work products (primarily, the SA diagrams) occurs almost continually. Each diagram is inspected by a 'commentor,' who is a person with substantial knowledge of the particular subject matter. The commentor must *write* the comments on the diagram, and the author must reply in writing on the document. So each diagram has an audit trail of its inspections, decisions, and revisions.

We were told that SA is most appropriate for larger projects, those involving at least 4 to 6 people for at least 6 to 9 months, and that the results become more impressive as project size and complexity increase. But once learned, it often is used on smaller projects. There is a substantial cost for acquiring the methodology and training everyone concerned in its use.

The diagrams

The methodology uses two basic types of diagrams—activity diagrams and data diagrams. On the activity diagrams, the boxes represent activities and the lines ('arrows,' in SADT terminology) connecting the boxes represent classes of data interfaces. Note that the diagrams are *not* flow charts and the arrows are not equivalent to the flow of data, as will be described. On the data diagrams, the boxes represent data classes and the arrows represent the activities that generate or use the data.

A general rule in the use of SA is: there must be no fewer than 3 and no more than 6 boxes on any diagram. Further, the methodology encourages readers and commentors to inspect the diagrams to make sure that this rule (and others) are being followed in practice. The reasoning behind the rule is that too narrow a view is being taken if only one or two boxes are shown, while more than six boxes will cover too much to comprehend. The diagrams are very 'information rich,' as we hope to show; even with a maximum of six boxes, they cover a lot of material.

The concept of *bounded context* is emphasized by SofTech. Each box, with all of its arrows, must completely describe its activity and nothing outside of that activity (the essence of decomposition).

Arrows entering the left side of an activity box represent data inputs, and those leaving from the right side represent data outputs. Arrows entering the top of a box are control, and those entering the bottom represent mechanisms.

Control arrows must always be present. They indicate the conditions or constraints under which an activity is performed. To illustrate, a batch control total is an example of a control signal that enters the validation activity; it helps determine if the batch, as received, is complete.

Mechanism arrows are used much less frequently. They relate to the resources that are used to perform the activities; a person or a computer are examples.

An output from one box can be a control or input to another box on the diagram. One box may represent the computation and checking of batch control totals, for instance, and one of its outputs would be a control signal to another box where the validation of individual transactions is done. If a control total did not check out, then the batch of transactions would not be passed on.

Parallel processing is allowed and can be shown on an activity diagram. Also, necessary sequence is shown. For instance, a batch control total must be confirmed and the individual transactions must be validated before the batch of transactions can be passed on for further processing.

Activity model. The top-level activity diagram has from three to six boxes that represent the complete system being investigated—all of that system and nothing but that system, as it interfaces its environment. This may not seem to be very complex, but since the boundaries of these top boxes are retained throughout the decomposition, it is an important step. And particularly in the case of a *new* system, creating this diagram can require a good amount of time. It is quite possible, we have been told, for the job to require several man-months to determine the best three to six major components of a complex new system. The decomposition may involve the analyst in questions of organizational changes, re-assignment of responsibilities, possibly enlarging the scope of the system beyond what was originally thought, and so on. *This* is the true structured analysis, not drawing the diagrams; the diagrams just support the effort.

Each box has a three to six word description of the activity: 'pass valid data,' and 'try to fix' are examples. Further, each arrow is labelled: 'delivery data,' 'batch controls,' and 'kind and number of errors' are examples.

Next, these boxes on the top-level diagram are decomposed, each on a separate diagram. These second level diagrams also have from three to six boxes each. Further, the inputs into a second level diagram, and the outputs from the diagram, come from the inputs, outputs, etc. of the box on the top-level diagram.

We may be giving the impression that creating an SADT diagram is a lot like drawing a flow chart. Such is not the case. Ross (Reference 2) gives a description of the mental processes required to create a diagram. First the box structure is built, by identifying the constraints that define the boundaries of each box—that is, what is considered to be inside the box. The use of 'control' arrows helps the author to state precisely his understanding of the activity. Next, the arrow structure, showing the controlling constraints for each box, is built. Then build the diagram structure, says Ross.

The concept of bounded context is important, both at the diagram level and at the box level. At the box level, it means the box and all of its arrows; the bounded context identifies all functions that are performed within the box, without regard to who performs those functions or how they are performed.

To illustrate bounded context, since it is so fundamental to SA, consider a central billing subsystem within a larger order-shipping-billing system. The central billing box on the top-level diagram might show (a) shipping documents as input, (b) data file additions, changes and deletions, plus inquiries and corrections as control, and (c) reports, invoices, and other inventory and sales data as output. In such a case, the author has determined that the maintenance of the customer billing data file is, in his mind, a part of this function. Eventually, he will explicitly define just what data and what data file maintenance are to be included in this function. The author has decided that the maintenance of the accounts receivable data and the posting of customer payments are *not* a part of this function, while the correction of errors on invoices *is* a part of the function. Later, as decompostion proceeds (or based upon the comments of a commentor), the author may conclude that defective merchandise returned from customers and the consequent issuance of credit invoices should also be a part of this function. If so, he goes back to the top-level diagram, as well as the appropriate lower level diagrams, and makes the changes. Eventually, he will feel that he has fully defined this function and that he has identified all of the inputs, outputs, and controls that apply.

We give this example to show how the SA diagrams help the author come to grips with stating the requirements. The diagrams are not used after the fact, to record the requirements; rather, they are very much a part of the process of developing the requirements.

Arrows may branch to two or more boxes. By labelling, the author can indicate whether all the data interfaces with all of the boxes or whether each box uses only part of the data.

An activity box becomes 'active' when it uses some inputs and control to produce some output. The complete story of a diagram includes all of the ways that all the boxes can become active. Thus an SA diagram is very information-rich. One diagram conveys about the same amount of information as 10 to 15 pages of text, we were told—and, in one instance, some 2,000 pages of text for the requirements of a military information system were reduced to about 40 SA diagrams.

The process of decomposing the activity diagrams is continued until no further decompostion is warranted.

Data model. After developing the activity model, the author begins developing a top-down decompostion of the data, by drawing successive levels of a data model. Data classes on the top-level diagram might include, for instance, the data definitions, the data files, transactions, and reports.

The data model is *not* a mirror image of the activity model, we were told. The boxes represent the data classes (at that level of decomposition) and the arrows represent the activities that inter-relate the different data classes. But the data model gives a quite different view of the system. Typically, the author gets new insights into the requirements and must go back and revise the activity model.

In SofTech's public papers on SADT, not much information is provided on the development of data models.

Participants. An 'author' creates the diagrams, as discussed above. A 'reader' reads one or more diagrams to gain understanding. A 'commentor' is an expert in the subject area who acts as an inspector *of content*; he reads and enters comments and questions in writing on the diagram, as has been mentioned; also, there may be more than one commentor for a diagram. A 'technical committee' resolves technical issues. A 'project librarian' maintains the file of the various model diagrams and generations of those diagrams. An 'instructor' trains authors, readers, commentors, librarians, data procesing managers, and user department managers in the methodology, to the level of detail appropriate to their jobs. And a 'counselor' is an expert in SADT who acts as backup for the teacher during the first project.

Example of use

Combelic (Reference 3) gives a brief description of the use of the SA portion of SADT at ITT Europe. The company adopted this SA portion in early 1974 and, in mid-1975, began developing its own compatible structured design methodology for real-time communications switching software. What the company is now using is a combination of the SA portion of SADT plus the inhouse design methodology.

The primary inputs to the SA function, says Combelic, are the list of customer requirements plus the functional specifications of the telephone hardware of the system. The output of the SA function is a functional requirements model—the set of activity diagrams, many of which are accompanied by a page or two of explanatory text. The functional requirements model emphasizes the *what* of the system, not the *how*.

SA provides a disciplined way of understanding requirements in detail before starting design, says Combelic. Further, the method promotes teamwork, in an environment where a team might consist of people with widely varying experience, from up to eight different ITT companies, speaking six different languages.

ITT Europe has found the following benefits from the use of SA. There has been a decrease in overall software development cost of at least 20%. The quality of the software has improved; there has been a reduction in the number of bugs found during integration testing by a factor of between two and ten. The method has forced high level decisions early, providing a sound basis for later lower level decisions. It provides an essentially continuous review and inspection by means of the written comments (in effect, continuous 'walk-throughs'). It encourages agreement on the requirements before beginning design. It has allowed non-software people to better understand the contribution of software to the operation of the system being built. It has provided an easy way to measure progress during the analysis phase. And it makes staff competence and incompetence visible.

There have been some problems and mistakes in the use of the methodology, says Combelic. It is hard for authors to always think in purely functional terms. At first, they were bothered by the lack of a design methodology to accompany SA, so they tended to want to think of *how* along with *what*. The method was oversold at first, as a panacea. The company found that mere 'training' was not enough; 'education' was required to accomplish the needed fundamental change in mental outlook. Potential authors must be selected on the basis of intelligence and willingness to try the method, rather than just on experience. And, he says, the training cost is substantial—\$20,000 to \$40,000 per course, for up to ten authors (but worth it).

In his verbal presentation at the 1978 National Computer Conference, Combelic mentioned three main problems with the methodology that still exist (and, by implication, that probably exist with other similar methodologies). These are: there are no formal criteria for guiding decomposition, there is no way to enforce semantic rigor, and there is no way to judge if a diagram is 'good'—or even to know what 'good' means.

Combelic stresses a point that seems to apply to all top-down analysis and design methodologies. That is, the method takes much more time before design starts than the methods they used previously. This time is spent in gaining a thorough understanding of the requirements. But it does cause impatience among some participants and management.

(On the question of acceptance, a leading consultant in the field, who teaches another structured analysis and design methodology, has been quoted as saying "Our best customers of our structured design method, who started using it two years ago, are no longer using it. The reason is that the *users' representatives* had not been trained in the new methodology. It did not meet their expectations so they could not accept it." So training must include not only the analysts and programmers but also the data processing managers and the user department managers, so that they understand why the method takes more time at the outset.)

Combelic makes another interesting point. About two-thirds of the value of the total analysis and design methodology is in the SA part, he says. That is where the biggest gains are realized. But the SA part, by itself, is not sufficient; the design method is essential. If the SA authors know that the design points will be considered, they are more willing to defer those points to the design phases and concentrate on the *what* during the analysis phase.

For more information on SADT, including discussions of other user experiences, see Reference 4.

IA—Information Analysis

We recently attended a seminar in Holland, presented by the IFIP Applied Information Processing Group (IFIP/IAG), on Information Analysis (IA). This methodology was developed by Professors B. Langefors, M. Lundeberg, and their colleagues at the University of Stockholm, Sweden, and they presented the bulk of the seminar. We had heard many comments about this methodology over the past several years, and welcomed this chance to learn about it first hand.

The overall methodology consists of five phases: change analysis, activity studies, information analysis, data system design, and equipment adaption. The last two phases are concerned with design, and so we will touch on them only briefly in this discussion. We will concentrate instead on activity studies and information analysis. But first, a brief description of change analysis.

Change analysis. The first step in any information system study, say the IA developers, should be an identification of the underlying problems, from a management or organizational standpoint. When a new information system is being considered for some part of an enterprise, review that part of the enterprise and try to identify the types of changes and improvements that are needed. It may turn out that changes are needed in the 'object' system, that handles the physical materials as well as the information.

In short, try to see the new information system in its complete context. What are the basic problems that management wants to solve? What seem to be the causes of those problems? What should be the goals of the project?

This concern with the 'object' system, covering physical products as well as information, carries over into the next phase, activity studies.

Activity studies

We can describe these studies best by outlining the mechanics of performing them. The mechanics are quite similar to what is used in the remaining three phases of the overall methodology.

The method emphasizes user participation in the process. Thus, the 'analyst' function generally involves both system analysts and users.

The method involves the use of a very simple form—a large, empty square, about 7 inches on a side, drawn on an 8 1/2 by 11 inch sheet of paper. The square defines the boundaries of what is being analyzed or designed. Everything within the square is considered to be a part of the function being studied, and nothing outside of the square would be a part of the function. So the concept of 'bounds' is injected at the outset.

The analyst draws small boxes just outside the top of the square to represent documents or materials flowing into the function from the 'outside world' or from other functions. Management policies and guidelines, appropriate to this level, also show as inputs. Similarly, boxes are drawn just outside the bottom of the square to represent documents or materials flowing out of the function.

The analyst analyzes the function by identifying from three to six activities that make up the function. Note that these probably will involve both information handling and materials handling activities. For an order-production-shipping-billing function, the activities might include customer order processing, invoicing, production, and inventory handling and distribution, as an example. In this instance, the analyst sees the overall function as consisting of these four activities.

To diagram this function, the analyst begins by identifying the inputs and outputs and drawing the appropriate boxes at the top and bottom of the square. Then comes a bit of a surprise. For the four activities, the analyst just makes four large dots within the square, well separated from each other, and beside each one writes its name.

A *dot* for an activity? Yep, that's all—and for a very good reason, we think. The analyst cannot write anything *inside* the dot; only the name can be written beside it. The method is saying to the analyst, "Do not think about the details of this activity yet, just think about it as a yet-to-be-defined activity." The method is thus forcing the analyst to use a 'levels of abstraction' approach.

Next, the analyst draws a line from one or more input boxes to one or more activity dots, as appropriate. In our example, customer orders flow into customer order processing, and raw materials flow into production. Similarly, the analyst draws lines from the activities to the output boxes. Thus, a line is drawn from invoicing to invoices, and from inventory handling and distribution to products at customers' sites.

This may sound like a simple process but it is not. It takes quite a bit of thought to properly define the boundaries of the object system. We tried it and discovered a tendency to set the boundaries too large. And the incorporation of the materials handling activities comes as a surprise to analysts accustomed to working with information only.

Remember that, in the change analysis phase, the basic problems with the object system were (hopefully) identified, and the goals being sought in the new system were identified. In defining the scope of the object system, the analyst must keep these problems and goals in mind. They help determine the scope.

Next, the analyst looks at each activity (dot) within the square. He must determine what major types of information and materials are produced by each activity. Information and/or materials flowing outside of the function involve lines drawn to boxes at the bottom of the square, as already mentioned. But for information and/or materials flowing to another activity (or activities) *inside* the square, not only must lines be drawn but also boxes are drawn and labelled. Thus 'approved customer orders' might flow from customer order processing to both invoicing and to inventory handling and distribution.

In short, this top-level diagram might have five or six information and material input boxes at the top, five or six information and material output boxes at the bottom, the four activity dots, and perhaps four information type boxes for internal information types, all connected by appropriate lines. The diagram, when completed, is assigned a control number.

The diagram is basically simple to draw. There is not a lot of writing, just the labelling of the dots and boxes. The analyst must concentrate on *what* is being done within the scope of the function, and not *how* it is done. There is no way to indicate the procedures used within an activity. Only necessary sequence is indicated; customer orders are checked before being passed on, but no effort is made to indicate whether the approved orders go first to invoicing or to inventory handling.

The next step in the process is to analyze each of the activities on the top-level diagram. The analyst uses another blank form with the empty square on it. From the top-level diagram, he determines what inputs flow into the selected activity, and draws a box for each at the top of the square. Similarly, the outputs are determined from the top-level diagram and boxes are drawn at the bottom.

Now the analyst must determine what component activities make up the selected activity. For customer order processing, these might be credit check, order approval, and adjustment handling. In this case, three dots are put inside the square and connected to the appropriate inputs and outputs. Then any inter-activity messages or material flows are identified and drawn in.

As this process of decomposition continues, it may become evident that the materials handling activities need to be changed, in order to meet the goals of the project. Such a redesign probably will require the services of other specialists.

Also, as this process of decomposition continues, diagrams are developed that have no materials handling activities on them; they consist solely of information handling. Having reached this point, the analyst is ready to begin *information analysis*.

A point worth noting is that at least the three users that addressed the seminar tended to skip over this activity analysis and instead go directly to information analysis. But IA's developers feel that system quality will improve, and management will be better satisfied with results, if change analysis and activity analysis are done.

Information analysis

The information analysis phase follows much the same procedure as activity analysis, except that only information processes are considered, not materials handling activities. The same basic form—an empty square on a sheet of paper—is used.

Each information handling activity from the appropriate activity diagram is analyzed on a separate information flow diagram. The input information sets, as determined from the activity diagram, are drawn as boxes just outside the top of the square. The output information sets result in boxes at the bottom of the square. And now the analyst must identify the information processes needed to transform the inputs into outputs, as well as any *necessary sequence* in those processes. Thus the diagram must show what different types of information *must* be available for processing in order to produce an output. In our example, a customer order is needed before order entry can be performed. Order entry and customer credit data are needed before the credit check can be done. Also, if in drawing the diagram, a sequence appears to exist but in fact is not really necessary, that point is indicated on the diagram. Further, intermediate information sets are indicated by boxes between the information processes, suitably labelled.

The next step is to decompose the information processes from such a diagram, each one diagrammed on a separate form. The diagram should show any information sets that exist in different generations (such as the updating of the customer file). Note that we are still concerned with analyzing requirements, not with the design of the new system.

Having continued this decomposition as far as practical, the analyst next performs a *data analysis*. Each information set is broken down by showing a listing of the data types that make it up. This analysis results in a hierarchy of data types. For example, customer orders might be sub-divided into the main order types (single orders, bulk orders, customer returns for adjustment, etc.). Eventually, this breakdown lists the data fields that make up a given type of data record.

Then the analyst develops a list of the information processes that have been identified. For each process, a process table is created, showing what inputs are needed, what calculations must be peformed, and what outputs are produced. Note that this is the first time that detailed procedures (the calculations) have been considered. The 'levels of abstraction' approach used by IA has delayed this consideration as long as possible.

We do not want to give the impression that IA is a once-through, top-to-bottom methodology. At each stage of decomposition, the analyst may discover something that causes him to go back and revise some of the higher level diagrams. The same thing can occur with the data analysis. In the process of decomposing the data types, it may become apparent that the information flow diagrams have to be changed somewhat.

The next two phases involve the design of the new system and its adaptation to fit particular equipment. Since these subjects are outside the scope of our discussion, we will treat them only very briefly.

Using the results of the information process analysis and data analysis, the system designer(s) develops an (almost) equipment-independent solution. Design diagrams are used that are very similar to the ones discussed above. Instead of processes, computer programs are identified in general terms, such as sort, update, print, and so on. Following this, a program-oriented data structure is developed, for the files used by each program. Then Michael Jackson's approach to program structure is suggested, with the structure based on this data structure.

Having developed an equipment-independent solution, the final step in design is to adapt the solution to fit particular equipment. Also, practical considerations, such as controls, audit trails, and backup, are brought in.

These are the main characteristics, then, of Information Analysis. But what have been the experiences of users? Three users described their experiences at the seminar mentioned above.

Some user experiences

Desisco Nederland B.V. provides consulting services as well as application system development services. In 1972, Professors Langefors and Lundeberg gave a seminar in Holland on IA that was attended by a Desisco representative. This person started using IA on a client program he was working on that was already in the problem analysis phase (beyond the feasibility study phase).

Very soon, the IA diagrams showed that the scope of the project was much larger than had been indicated in the original requirements statement, developed by the client during a feasibility study. The client agreed that the scope was indeed larger than had been estimated. Since this was all too characteristic of such projects, Desisco management was impressed.

The method is now used by all Desisco personnel for feasibility studies, information system analyses, and system design. They emphasize its use in the *early* stages of a project, whether a new system or the re-design of an existing system. It is very helpful for establishing the goals for a new system, for determining users' information needs, and for communicating between users and developers, they find.

Perhaps the best way to start a project is with the whole object system, or a large part of it, and perform a change analysis before doing the information analysis, say the people at Desisco. In such a case, the analyst deals with higher levels of management and is more likely to identify the real problems.

If the project team is not authorized to study the whole object system, then Desisco recommends that another approach be used. That is, do not stop at the designated inputs and outputs for the information system under study, as specified on the top-level diagram. Instead, use information analysis diagrams to trace the inputs back into other systems and the outputs to other systems. Make sure no gaps or overlaps occur between the new system and the others.

The second user experience applied to a large publisher of newspapers and magazines in Holland, as described by a project team member from outside that organization. This company had set up a good sized (3 to 4 man-year) pilot project in 1975 to try out some of IBM's IPTs for developing an on-line interactive database system. Among other things, the team used HIPO charts and functional decomposition. Within six months, the two analyst/designers assigned to the project stopped it; they were unhappy with the results they were getting, particularly from the use of HIPO. After some re-thinking, they decided to start over again, this time using IA.

The team did not use activity analysis, but did use information analysis much as described above. Data analysis was performed in a somewhat different manner (using the 'third normal form' approach of relational databases). Since a database would be involved, particular attention was paid to the relationships among the data items. And when the processes were decomposed into procedures, Nassi-Schneiderman charts were used rather than text or pseudo-code.

Nine months later, the analysis had been completed and design began. The system became operational at the end of 1977. Moreover, management considers both the use of IA and the resulting system as successful, so the use of IA at this company is expected to grow.

In the third case, the Swedish National Central Bureau for Statistics, in Stockholm, was concerned about providing better services to users and decided to use IA to help achieve this. The Bureau has large statistical data files, and it soon became apparent that they could not define 'user activities' in the use of those files. So activity analysis and information analysis were not too useful for them.

However, they did find much value in what they call 'object system' analysis and developed a structure diagramming method to show the relationship between real-world items for their data files. The method is quite similar to the well-known Bachman data structure diagrams, except that, in addition to one-to-one and one-to-many relationships, many-to-many relationships are also shown.

The result has been the development of data structures that 'model reality' much better than the old files did. So the Bureau is finding that users are better satisfied with their services.

For more information on Information Analysis, see Reference 5.

Analyzing user needs

Having discussed HIPO (briefly), SA, and IA, let us now 'step back a pace' and analyze what has been said. Based on the structure of these methodologies and the user reactions to them, what seem to be the most desirable features of an analysis methodology?

Any such generalizations about a desirable methodology admittedly would be based on a small sample of user experiences. But until there is a larger user population for these methodologies, only a small sample is available. The generalizations can provide a hypothesis that can be validated (or not) as more use is made of these methodologies.

The first desirable feature, recommended by the developers of both SADT and IA, is controversial. That feature is 'identifying the basic problem,' which the IA people call 'change analysis.' But the users of IA who addressed the seminar generally omitted this step, and instead tended to go directly into information analysis. Let us show the important role this step can play, adapted from an actual case described at the seminar.

Identify the problem. Consider the case of the data processing department that has been working on the preliminary plans for an on-line interactive order-shipping-billing system. That is, assume that these functions are currently being done on a batch system. Someone has conceived the idea that system performance could be improved if orders were entered immediately, rather than held up for batch accumulation. So preliminary plans are developed showing how an on-line interactive system might work and what the time savings might be. And now these preliminary plans are being presented to various members of management.

Assume further that the products shipped by this company are perishable (for example, processed dairy products) and that customers can return unsold products for partial refund. However, the company cannot return anything to its suppliers for partial refunds. It bears the loss for all over-age products.

During the presentation of the preliminary plans, the speed of entering customer orders is emphasized. But one sharp manager speaks up and says, "That is all very well, but the system is not addressing our *real* problem. Our real problem is that our customers make mistakes in their order quantities, ordering either too much or too little. If they order too much, we end up giving them partial refunds. If they order too little, they and we lose sales. Both of these situations are wasteful. Your new order entry system does nothing to correct this situation."

This comment might trigger off another from a second manager. "Yes, the same sort of problem occurs with our suppliers. Since we do not know how much our customers will order, we tend to order too much or too little raw materials from our suppliers. If we order too much, we end up with waste. If we order too little, we either cannot supply some customers or we pay more for expedited orders. The new system will do nothing to solve this."

Out of such a discussion might come the idea of trying to forecast the order quantities for individual products by individual customers, based on the main factors that influence customer orders (perhaps holidays, vacation periods, special events, weather, etc.). Then provide these forecasted quantities to the customers on their order forms and let them revise the quantities as they see fit. Hopefully, in the majority of cases, no revision will be made. When revisions are made, again hopefully they will not be large. So the company will have a better idea of how much the customers will order and in turn will know better how much to order from suppliers. As a result, the boundary of the system has been extended from order entry out to the customers' ordering decisions.

In this example, the basic problem was identified almost by happenstance. Instead of depending on chance, though, perform a 'change analysis,' say the IA people, and actively search out the basic problems.

But now on to the characteristics that the users seemed to agree were most desirable.

Desired characteristics

Levels of abstraction. Decomposition by levels of abstraction appears to be fundamental, for analyzing complex systems. With this approach, the analyst is concerned with just one level of detail at a time; all consideration of lower levels of detail is postponed. So the analyst is more concerned with breadth of consideration than with depth, at any level.

It is not realistic to assume that all analysis will be fully top-down, in the levels of abstraction approach. For one thing, there will be a good amount of iteration, where the analyst has to change something at a higher level because of some just-discovered lower level factor. Then, too, the analyst may want to analyze a particularly complex detailed aspect at the outset, to see what it really consists of, rather than approach it top down.

Three to six elements. Coupled with the levels of abstraction approach is the idea of limiting the number of elements being considered at any one time to between three and six. This restriction forces the analyst to aggregate similar things and/or separate things with differences—so the analyst must look for similarities and differences, which increases understanding.

By looking at three to six elements, the analyst is also forced to consider the relationships among these elements. This also increases understanding, as compared with looking at only one element at a time.

Bounded context. This concept requires that the analyst identify all processes, data types, and relationships among them within the specific group of elements being analyzed. All that belong must be included; all that do not belong must be excluded. This precise definition of boundaries also helps promote understanding.

Analyze both activities and data. The tendency is to analyze the activities and to treat the data almost as an after-thought. But the methodology should require that the data types be analyzed top-down in much the same manner as the activities are analyzed.

For instance, in the order-shipping-billing example for processed dairy products, given above, the analyst might, on the top level diagram, only show something like 'customer reorder data.' During the data analysis, this would have to be broken down into its constituent elements. It might become apparent to the analyst that a data file of the dates of holidays and special events might be needed, if these data types were part of the customer reorder decision. Eventually, the analyst would have to determine just what data would be supplied to the customer, to indicate the basis upon which the forecast has been based. Each of these steps might give the analyst more insight into the application and point out the need to revise the activity diagrams.

Graphical notation. This also might be a controversial characteristic, although HIPO, SA, and IA all use it. But a graphical language can be both relatively simple and quite powerful in its ability to convey information. Because of its pictorial nature, it can show boundaries, activities, data types, relationships, and the number of elements within the boundaries. Thus, it too can aid comprehension. In fact, it may be the *only* effective way to get a group of users to look at a requirements document.

Simple for users to grasp. An important factor of the methodology should be that it provides an effective communication bridge between the analyst and the user and between the analyst and the designer/developer. A graphical notation, with simple, uncluttered diagrams, can provide this. Users have been able to readily 'grasp the message' with the three graphical methods discussed in this report. And in two of them—SA and IA—the necessary flow of data within the boundaries is also easily grasped.

Easy to change. As we have tried to indicate, the diagrams go through many revisions during the analysis phase. So it is important that they not be difficult to redraw. If they are simple combinations of lines and boxes, with very few words, they will meet this objective. (We will say something about automated versions shortly.)

Numbering system. Closely related to this characteristic of ease of change is the system of numbering the different diagrams—as well as the different generations of each diagram.

It is not unusual for an analyst or designer to suddenly see a 'neat' way to do something or to express something. Before he or she plunges ahead with this neat solution, it pays to check back to prior generations of the same document. On one of those documents might well be the evidence that this neat idea was considered once before but had to be rejected for a legitimate reason. Thus getting at prior generations of a diagram is important, and a good numbering system will help accomplish this.

Defined procedure of use. The methodology should have a debugged, prescribed method of use—including suggestions on how to decompose systems. One reason, of course, is that the various types of users (analysts, designers, programmers, managers, etc.) all should have the same understanding of the use, to the level of detail that their jobs require. But there is another important reason. There is a tendency for analysts, designers, and programmers to modify a methodology to suit their particular whims. If this is allowed to happen, soon there will be a reduced compatibility among the diagrams. So users should be trained in the correct use of the methodology and then the use should be monitored.

Adequate training material. As indicated, there will be a variety of types of users—analysts, designers, programmers, data processing managers, user department managers, project librarians, and inspectors—so the availability of good training material is essential.

Frequent inspections. These should be a basic part of the methodology. One type of inspection is for content, to make sure that the requirements are being fully and accurately stated. Such inspections are best performed by people who know the application in depth. Another type of inspection is for correct use of the methodology, best done by people who know the methodology in detail.

It can be very helpful, we gather, if these inspectors write their comments and questions on the diagrams and the analysts also write their answers on the diagrams. The diagrams thus will provide an audit trail of their evolution.

Leads into design method. Users of top-down methodologies tell us how hard it is for analysts and designers to change their way of thinking, which has typically been bottom-up, to top-down. And it is just as hard for them to concentrate on *what* the system must do, during the analysis phase, and to ignore the *how*. If the analysis method just naturally flows into the design method, they are more willing to delay consideration of the *how*. So the presence of a design methodology, and one that is compatible with the analysis method, is important. At the same time, the method should not force the designer to use the requirements structure in the design. There should be a natural way to move from the structure for displaying requirements to the design structure for the new system.

From the seminar presentations and our talks with users of analysis methodologies, we gather that the above characteristics are the ones that have been found most beneficial.

Why is HIPO controversial?

As we indicated earlier in the report, IBM's HIPO is quite widely known and thus provides a basis for comparison with other methods.

But from our contacts and discussions, it appears that HIPO may be the least used of IBM's IPTs. Yes, we have heard reports of satisfied users, but more often it seems to be a case of "we tried it and didn't like it." HIPO seems to be receiving more than its share of criticism.

Why is this the case?

If the above list of desired characteristics is valid, then it can provide an answer to this question. HIPO has very few of these characteristics.

It does have a good degree of bounded context, for instance; it uses a graphical notation that has proved to be readily grasped by users and developers alike. When coupled with structured walkthroughs (another IPT), it provides for a good inspection mechanism. But that is about as far as HIPO goes.

What HIPO does *not* provide is more revealing. It does not really use the levels of abstraction approach. It creates a hierarchical chart of functions and then the analyst examines one process at a time; it does not show from three to six processes on an IPO chart, along with the relationships among them. Each process includes detailed procedures, almost at the pseudo-code level; for other than bottom level boxes, these are usually only control procedures. HIPO does not provide for a data analysis that parallels the activity analysis. The IPO charts are not particularly easy to redraw because of their wordiness. No disciplined method of use exists. The requirements structure (the hierarchy chart) tends to be imposed upon design. And HIPO does not lead naturally into a compatible design methodology, we gather from users, so analysts may be reluctant to postpone design considerations and concentrate on what the system should do, during analysis.

Perhaps this lack of these characteristics—characteristics that some users feel are important—accounts for HIPO's failure to gain wider acceptance.

The analyst's 'work bench'

The term 'work bench' is being applied to a set of software tools that help the system developer do his or her job. Initially, the concept was applied to the programming function. It is now being extended to support the functions of system analysis, system design, and data administration.

To see how the analyst's work bench might operate, assume that each analyst has a graphics terminal tied to a computer with the necessary support software. Also, means must be provided within the department for printing out graphic diagrams, perhaps via one of the new photocopy-like printers. The work bench might then be expected to perform the following functions.

Perform routine tasks. The work bench should relieve the analyst of routine duties, so as to concentrate on the application. The work bench should allow the analyst to draw graphic diagrams on the terminal, perhaps by the use of a light pen, and enter text via a keyboard. The system should apply diagram identification numbers. Diagrams would then be printed out, for review by commentors. Commentors could either write their comments on the paper diagrams (and someone else enter them into the system) or could enter them directly on a terminal. The work bench should allow the analyst to easily correct the diagrams and create updated versions, filing the old versions away. It should enforce the use of standard procedures, should retrieve old generations of documents for perusal upon demand, and should provide the 'Help' facility to tell the analyst what options are available at any decision point.

Support analysis. In addition, the work bench should help the analyst perform the analysis of user needs. One way to do this is to prompt the analyst by means of a checklist, developed from several sources: from experience, from the results of inspections, from published checklists developed by others, and so on. The work bench might help detect missing, incomplete, and/or inconsistent requirements statements, which are prime causes of system difficulties.

We think you will be seeing and hearing a lot more about system development work benches in the months just ahead.

Next month, we will continue our discussion of new software development methodologies, picking up at the point where user requirements have been defined.

REFERENCES

1. Jones, M.N., "HIPO for developing specifications," *Datamation* (1801 S. La Cienega Blvd., Los Angeles, Calif. 90035); March 1976; p. 112 ff.
2. A series of papers on the analysis of user needs and determining requirements, included in *Software Engineering* (IEEE Computer Society, 5855 Naples Plaza, Suite 301, Long Beach, Calif. 90803); January 1977, p. 16-34; price $10.
3. Combelic, Donn, "Experiences with SADT," *Proceedings of 1978 National Computer Conference* (AFIPS Press, 210 Summit Avenue, Montvale, N.J. 07645); p. 631-3; price $60.
4. For more information on SADT, including some user experiences, write SofTech, Inc., 460 Totten Pond Road, Waltham, Mass. 02154.
5. For more information on Information Analysis, including two new books on the method (in English), write Research Group ISAC, Department of Information Processing, University of Stockholm, S-106 91 Stockholm, Sweden.
6. For a copy of the IAG seminar handout material on Information Analysis, write IFIP/IAG, 40 Paulus Potterstraat, 1071 DB Amsterdam, Netherlands; price Dfl. 50 (via surface mail).