EDP ANALYZER

Editor and Publisher: RICHARD G. CANNING

EDP ANALYZER

SEPTEMBER, 1981
VOL. 19, NO. 9

DEVELOPING SYSTEMS BY PROTOTYPING

In data processing, unlike in other fields such as engineering, the first working version of a program often becomes the final operational version. We have heard people in the field lament that the computer field really should become more like the other fields and make software prototypes first—and then develop operational systems from these. One benefit of this approach is that such systems would probably require much less maintenance, because there would be fewer design errors and oversights. The good news is that software prototyping is now feasible and cost effective, due to some fairly new tools.

American Warranty Company is a vehicle service contract administrator. The company handles the paperwork, co-ordinating functions, and actuarial studies relating to extended warranty contracts on new cars and trucks, thus relieving both the automobile dealers and the insurance companies of this work.

American Warranty has regional offices in Georgia, Minnesota, Washington D.C., and Illinois; headquarters are in Los Angeles. American Warranty is servicing some 750,000 contracts (and they are adding roughly 180,000 a year) with about 40,000 claims a year against these contracts. All data processing work for handling the contracts and claims is performed at headquarters.

In mid-1979 American Warranty decided to bring their headquarters data processing work in-house. Previously they had been using a service bureau that had an IBM 360/50; their application programs were written in COBOL. After quite a bit of searching, they settled on a Wang VS 2200 system. This is a small business computer system, using a 16-bit mini-computer, with 512k bytes of memory.

With their new system, American Warranty obtained almost one billion bytes of on-line storage, 26 user work-stations, and five printers. Two of the printers are for word processsing and they produce typewriter-quality print. The data processing department has four people, all of whom are programmers with previous COBOL experience.

ISSN 0012-7523. Multiple copy prices listed on last page.

One reason American Warranty chose the Wang system was because they could move their application programs and their data files from the IBM system to the Wang easily. Wang provides an IBM-to-Wang COBOL source code converter and an EBCDIC-to-ASCII data code converter for file conversion. Since the company was moving from a batch environment to an on-line environment, they chose to convert about one-third of their programs and re-write the other two-thirds.

The second main reason American Warranty chose Wang was because of its versatility and power. Wang provides a program generator for creating COBOL, Basic, or assembly language programs, plus a screen editor, a data dictionary (which Wang calls a control file), a report generator, a menu generator, and several other utilities. Each of the utilities creates COBOL source code which can be modified or, more typically, enhanced with manually written subroutines. The company finds that for most of their applications, they need to manually write code only for complex data validation and mathematical calculations.

American Warranty is using these utilities to create applications through an iterative, prototype-like approach to development, with a lot of end user participation. For example, when their purchasing system was being developed, the purchasing manager was interested in being involved in the design and development. With the Wang system, the data processing department was able to get him actively involved. They began by asking him to design some output reports that his department would need—first on paper and then on the Wang. It took a data processing staff member about one hour to show the manager how to use the Wang screen editor to design his reports. After several iterations, the manager was satisfied with these report formats. Then a programmer created the necessary detailed data definitions and their validation criteria using the Wang data dictionary. At this point they had some purchasing report formats and a database with which to run the reports. This phase of the development process took two weeks.

Next, some work-stations were installed in the purchasing department and the employees were shown how to generate these reports. They were encouraged to study the information provided by the system and recommend any changes they felt were needed. The data processing department used these comments to create several improved versions of the reports and the database, again using the Wang facilities.

Meanwhile, data processing began work on extending the system by creating some inquiry and retrieval programs for use by management and employees in other departments. When this phase was completed, these users were also taught how to use the system and encouraged to experiment with it. Their recommendations were also used to created improved versions of the prototype.

In this way, the purchasing system was developed—modularly and with a great deal of user evaluation. And although the data processing department made it clear that each version was not the final production system, these prototypes were not taken away from the users—each was used as if it were the production version. When the system appeared to be complete, that final prototype did become the production version; no re-coding was necessary. This final prototype contains some 13,000 lines of code, most of which was produced by the generator, and the entire process took six weeks to complete.

In all, American Warranty is pleased with the Wang system. Its several programming aids has made it possible for them to develop systems in an evolutionary manner, with much end user involvement in design and evaluation.

RCA Corporation

RCA Corporation is a multi-national organization comprised of seventeen independent operating companies. These are leading companies in the television, broadcasting, vehicle rental, floor coverings, record, video disk, and other fields. RCA, with headquarters in New York City, is ranked as the thirty-sixth largest industrial corporation in the U.S. by *Fortune* magazine. In 1980 they had sales of over $8 billion, and employed some 133,000 people worldwide.

Each of the operating companies has its own data processing function. In addition, corporate data processing has some multi-division systems—payroll, finance, and personnel. The history of the corporate-wide personnel system is of interest to us here.

In 1974 top management decided that it was time to implement a corporate-wide personnel system. As Paul Berger and Franz Edelman describe in Reference 1, the corporate operations research group was consulted about creating an end user 'front-end' to such a system. The front-end would allow non-data processing people to query the system directly and generate reports from the database. Several of the corporate planners had used RAMIS through a time-sharing service, and they decided that such a front-end could be created for the personnel system using RAMIS.

RAMIS, from Mathematica Products Group in Princeton, New Jersey, is a database-oriented system that allows 'end user programming.' It provides data management functions, such as a report generator, interactive query facility, non-procedural language, data dictionary, restart and recovery features, plus select and sort features. RAMIS allowed the corporate planners to create applications for their own use without the need for professional data processing help.

The front-end project was to have two phases: a prototyping phase and a development phase. Development of the full system would begin only after management had seen the prototype of the front-end in use, and then approved the rest of the project.

Design of the prototype began in February and took five months to complete. The design team consisted of one experienced RAMIS user and two industrial relations managers, all of whom worked part-time on the project during this period. The front-end was developed on an outside time-sharing service using RAMIS. The process was iterative—the designers would create a prototype, let the users try it out, and then modify it based upon the users' suggestions. Each version was closer to the originally envisioned system. The final prototype consisted of the various data entry and inquiry 'screens' (what the user sees on the CRT screen) and the logic to link them together. And it contained a number of standard reports which users could easily request. Since RAMIS provided the data management functions, the designers only needed to define the contents of the files and the updating criteria. Actual operational data was supplied by one of the operating units.

The prototype was then demonstrated to RCA's top industrial relations executives. The executives were asked to bring to the meeting one real-life question they had encountered in their work the day before. One at a time, these queries were entered into the prototype system and the answers displayed for the executives to see. The system was not able to answer all of the queries, but it answered enough of them to convince the executives that the prototype represented a viable and useful user front-end system. Approval was granted to develop the full system.

Several important decisions were then made in mid-1974 about the final system. One was that the system should be modular, so that new functions could be created (via RAMIS on a prototype basis), then tested by users, and finally integrated with the rest of the system. The designers foresaw that the personnel system would require numerous enhancements over the years in order to comply with the increasing number of government regulations in the personnel area. They saw, too, that it would eventually handle all matters related to personnel information. And such has been the case.

Another decision was to divide the system into two parts—an interactive front-end portion and a batch updating portion (which they call the 'back-office'). The front-end would remain in RAMIS. Here again, the choice was made based on expected future changes. The designers believed that it would be much easier to modify the front-end using RAMIS than if it were programmed in a standard programming language, such as COBOL.

The batch portion, however, would be written in COBOL to increase execution efficiency; in this portion, the logic was expected not to change much. The updating of the database would occur during night-time batch updating runs.

Five months later, in December, 1974, the batch processing portion of the system was sufficiently complete to allow loading and begin maintaining the first division's database, for about 7500 employees. That division began using the system on a regular basis in April of 1975. Within the next twelve months, the system was moved from the time-sharing service to an in-house IBM 370/168 to operate under VM/CMS. During this period, four large RCA operating companies were added to the system, bringing the total database to 28,000 employee records.

A number of major sub-systems have been added to the system since its implementation in 1975. The company has added several benefit plans and an option for employees who choose to use a health maintenance organization rather than the company medical plan. A compensation budgeting and planning module and a quarterly EEO (equal employment opportunity) reporting capability have been added, and so on. So prototyping for new capabilities such as these is a continuing process with the personnel system.

By 1977 the system contained records for 55,000 employees, and it was handling close to one million transactions a year. As each new division was added to the system, the designers were concerned that the system might become overloaded and response times would deteriorate. But it handled the 1977 load just fine. The designers had feared that they might have to rewrite major portions of the system front-end in COBOL, to improve operating efficiency or user response time. However, this need has not materialized.

RCA currently has about 100 non-data processing employees accessing the system using RAMIS. The system has grown to two million transactions a year, with a database of over 130,000 employee records being maintained. New capabilities are still prototyped before being added to the production system. No major reprogramming has been necessary. And maintenance of the batch processing portion is handled by just one person. Following this successful experience, prototyping has become a common development methodology at RCA.

General Electric Company

General Electric Company is a multi-national corporation ranked as the thirteenth largest U.S. corporation by *Forbes* magazine, with sales of about $25 billion. The company manufactures a wide range of consumer and industrial products, with emphasis on electrical products, as well as data processing and data communication products. They employ over 405,000 people worldwide.

At the 1980 APL Conference, sponsored by I.P. Sharp Associates Ltd. of Toronto, Canada, Hassan Gomaa of General Electric and Douglas Scott of I.P. Sharp (Reference 2) described how a development team used the I.P. Sharp APL time-sharing service to perform software prototyping. The objective of the project was to more fully understand the diverse end user requirements for an automated system that the employees of a semi-conductor research and development laboratory would use. The development team believed that a working prototype would describe the user requirements better than written specifications could.

The development team designed the system for three types of end users—process engineers, equipment operators, and managers—each with quite different requirements. Process engineers, who design and refine the semiconductor production processes, would use the system to determine, through experimentation, the set of process steps that would provide the best production yield. They would enter the fabrication processes into the system through a dialog with the system.

The equipment operators in the various processing areas would use the system to receive the processing instructions for each batch of silicon wafers they were working on. They would also record the work they completed and the results of tests they performed. The instructions for each operator consist of ten to fifteen operations. Performing these operations may take from one-half hour to eight hours. A batch, or lot, of wafers passes through 60 to 100 such steps in the integrated circuit production process. Being a development facility, each lot of 25 to 50 wafers would require slightly differ-

ent processing instructions, so the entire operation is very complex. Finally, management would use the system to obtain results of product testing and to monitor the manufacturing operation.

Due to the complexity of the project, two people from General Electric and one from I.P. Sharp first wrote a preliminary set of specifications. From this I.P. Sharp began developing the prototype, using the I.P. Sharp APL time-sharing service, which is accessed by means of the I.P. Sharp network.

Several facilities on the APL service and network helped them to quickly create a prototype of the entire system. (1) APL allowed them to define the logic in the system fairly quickly. (2) The I.P. Sharp file system and its accompanying report generator made the creation and modification of files and reports fast and easy. (3) The I.P. Sharp system provided backup and recovery procedures so the team did not need to develop these programs. (4) The network service allowed managers to use and comment on the prototype at their convenience. The team created an online comment mechanism with which users could enter their comments, as well as review comments made by others, while they were using the prototype. These evaluators were located in New York State, Connecticut, California, and Toronto, Canada. Some of them even experimented with the prototype from terminals in their homes, by means of the time-sharing service. (5) The I.P. Sharp electronic mail system allowed the members of the development team to stay in daily contact with each other and resolve issues as they arose. The team members, being from two different companies, often were not at the same location. And (6) the time-sharing service allowed the team to develop the prototype even though the hardware that the system was designed to run on would not be available until nine months after the prototype was completed.

Once the initial prototype was created, all of the users were given a two-hour explanation and demonstration on how to use the system. The English-like query language and dialog procedures were explained. Then these users were given two weeks to experiment with the system. During this time, mis-understandings surfaced, ambiguities and inconsistencies were uncovered, and omissions were found. In some cases the users spotted features where they were not given all the information they needed, or the instructions were difficult or confusing to use. Also, the users even identified a few missing or incorrect requirements.

Based on the numerous comments made by the process engineers, operators, and managers, the team substantially revised the original specifications document and the prototype. From their experience they not only developed more true-to-life system specifications but they also learned how the final system, its files, and its data should be structured and which algorithms should be used.

The total prototype phase took seven work-months— six percent of their total estimated ten work-year development time. And the prototype cost less than ten percent of the estimated development costs (including the cost of the time-sharing service used in the prototyping process).

The main purpose of this particular prototype was to better understand how users would use the system, so the team concentrated on prototyping the interactive 'front-end' portion of the system that the users would see. They did not prototype the system's interfaces with other computers, for example, because they felt these requirements were more clearly defined.

What are software prototypes?

According to *Webster's 20th Century Dictionary*, the term 'prototype' has three possible meanings: (1) It is an original or model after which anything is formed. (2) It is the first thing or being of its kind. And (3) it is a pattern, an exemplar, or an archetype.

A. Milton Jenkins and J. David Naumann (Reference 3), in a paper on software prototyping, believe the second definition best fits the prototypes used in data processing because such prototypes are a first attempt at a design which generally is then extended and enhanced. Franz Edelman, in conversations with us, describes the process of software prototyping as "a quick and inexpensive process of developing and testing a

trial balloon." These two interpretations contain several important points.

First, the software prototype is a *live, working system*; it is not just an idea on paper. Therefore, it can be evaluated by the designer and/or the eventual end users through its use in an operational mode. It performs actual work; it does not just simulate that work.

Second, the prototype may become the *actual production system*, or it may be replaced by a conventionally-coded production system. Thus, the prototype may or may not be discarded.

Third, its purpose it to *test out assumptions*, about users' requirements, and/or a system design architecture, and/or perhaps even the logic of a program.

Fourth, it is a software system that is *created quickly*—often within hours, days, or weeks—rather than months or years. In the past, many people felt that software prototyping was impractical because it could not be performed quickly. With only conventional programming methods available, companies would not accept developing two versions of a system. So data processing departments had to create their production systems 'right' the first time around. But now some programmers are making innovative use of various types of software tools to get prototypes up and running quickly—tools such as the data management systems (DMS) that we have been describing for the past few months. (DMS generally involve the use of a database management system, plus retrieval facilities for queries and reports, selection and sorting features, and so on.) We will discuss several other types of tools later in this report, all of which can speed up the development process.

Fifth, the prototype is *relatively inexpensive to build*—meaning less expensive than coding in a conventional high level language. The reason the software tools make prototyping less expensive is that they create most, if not all, of the code.

Sixth, prototyping is *an iterative process.* It begins with a simple prototype that performs only a few of the basic functions in question. It is not required that these functions be performed elegantly or efficiently. But it is expected that, through use of the prototype, system designers or end users will discover new requirements and refinements which will then be incorporated in the next version. So it is a trial and error process—build a version of the prototype, use it, evaluate it, then revise it or start over on a new version, and so on. Each version performs more of the desired functions and in an increasingly efficient manner.

Uses of software prototyping

In our research for this issue, we found three uses of software prototypes: (1) to clarify user requirements, (2) to verify the feasibility of design, and (3) to create a final system.

To clarify user requirements

Using a software prototype to clarify user requirements is the most interesting, and most innovative, use we found. Daniel McCracken (Reference 4) points out that traditional, written specifications attempt to bridge the communication gap between the user and the designer. However, they do not serve this purpose well, because they are incomplete, they take a static view of requirements, and are often confusing. In addition, it is hard for an end user to visualize the eventual system from the specifications. Also, most users can not fully describe their current requirements—although they know what would be useful if they saw it—and even fewer users can identify their future needs, particularly for such things as tax law changes that have not yet been passed.

In place of the traditional process of determining user requirements, McCracken recommends creating a prototype to clarify the user's needs. First, the designer talks with the user about the needs that can be identified. Based on these discussions, and using (say) a DMS, the designer quickly creates a system that performs some functions that the user has requested.

Then, perhaps using a CRT terminal, the designer demonstrates the input screen formats and report formats the user would be using. And he shows the user the flow between the various elements in the proposed system: "If you select Option A on this menu, the system will respond with the following screen format." By creating a small file of data, the user can then use the sys-

tem to see how closely it fits the needs. Based on this actual use, most users will re-define and expand their requirements. Through several iterations of this process, the system comes closer and closer to what the end user wants. When the user is satisfied, the final version of the prototype can be used as the production system. Or it can be used to write the specification document, from which a production version can be developed using conventional development techniques and languages.

This process may seem longer than the traditional method of determining user requirements, but it is not. For small applications, the original prototype can be created in days—and then used and enhanced over the following few weeks. Obtaining the final system usually takes less than a month. For large, more complex, systems, this process might take a few months. We have heard estimates that prototypes for defining end user requirements require anywhere from 10% to 20% of the total estimated development time.

McCracken points out that the prototyping approach provides numerous benefits, particularly for the end user. For one thing, the specifications are more complete, because users can evaluate a prototype better than they can evaluate written specifications. Why? Because a prototype quickly brings users face-to-face with the types of problems they will face when operating the new system. For instance, an input or query screen format may look just fine on paper, but when used interactively the user may discover that it requires the entry of too much unnecessary information.

Prototyping also allows—and even encourages—users to change their minds about what they want, until they find a system that is truly useful to them. Numerous authors and people we talked with point out that when end users operate an automated system, it *always* changes their perception about what they really want. Prototyping allows this to occur early in the development cycle, when changes are cheaper and easier to make.

Traditional methods do not have a comparable experimentation phase; up until the end, the system is only a paper system in the eyes of the users. This point becomes even more important for large systems with many different types of users. With a working prototype, each user can evaluate the system from his own point of view and spot missing or mis-understood requirements.

The prototyping process thus eliminates the surprises that end users normally encounter at the end of the traditional development process, when they finally get to operate the system. Prototyping introduces that end-use step much earlier in the development cycle.

Prototyping appears to have benefits for the data processing department as well. Many proponents believe that prototyping end user requirements shortens the development cycle, because it eliminates most design errors—which are a major cause of missed deadlines and corrective maintenance work.

In fact, even enhancement maintenance can be reduced. Some of it is moved from the production stage of system use into the prototyping stage, in the form of iterations of the prototype. Also, if the prototype is saved, it can be used to test out future enhancements, to obtain more accurate user specifications on a continuing basis.

So prototypes are being used to clarify user requirements.

To verify the feasibility of a design

We have just described how prototyping can be used to better define the *external* design of a system—what the end user sees. Prototyping can also be used to verify the *internal* design of a system.

The most interesting use of prototyping the internal design that we have encountered was where a company wanted to integrate three purchased packages into one system. Prototyping allowed them to experiment with different ways of linking the packages. With the expected increase in the use of purchased packages, prototyping may become more widely used for this purpose.

In this instance, the designers first identified the functions that needed to be performed and where each function might be performed—in an in-house module or in one of the packages. A prototype was developed which showed the end users how the new system would operate (the external design). The prototype was then extended

to perform some of the linking of the packages (the internal design). While the efficiency of the prototype was not very good, it did allow both designers and users a chance to use the new system. Based on their reactions, a second prototype was developed, where efficiency *was* important. The prototype eventually evolved into the production system.

Another interesting use of prototyping we heard about occurred at an insurance company, where end users did the prototyping. In this instance, the actuaries used APL to create prototypes of the processing and calculation modules required for some new types of insurance. They created several iterations of the prototype, and when they were satisfied, they passed the program on to data processing to use to create the production system. When the production system was complete, the actuaries used their original prototype to verify the results produced by the production system.

A third example of prototyping to verify the feasibility of internal design involves creating a software prototype on a machine other than the final production machine. It may be that the production machine will not be available for some time, or the in-house machine is too busy to accommodate new development work. In any case, the designers can create a prototype on a different machine to verify the system design, and then create the final system on the production machine, when it becomes available.

Thus prototypes can be used to test the feasibility of design.

To create a final system

Think of prototyping as an evolutionary design methodology. It then follows that part (or all) of the final version of the prototype may become part of the production version.

For example, DMS are often used to prototype the interactive user portion of an application system. By the time the users are satisfied with the system, there may be no reason to re-write this portion for a production version; the DMS-based system may work efficiently enough. Jenkins and Naumann (Reference 3) point out that the only reason such a prototype should be reprogrammed is for economic reasons—where the expected savings in computer resources will be greater than the cost of re-programming. (However, these savings often cannot be actually achieved.)

Another consideration is that making changes to DMS applications is considerably easier than making comparable changes to applications programmed in a high level language such as COBOL. If the system is expected to change very much—and most systems do change considerably over time—then it might be wiser to leave some or all of the system in the DMS structure so that future maintenance will be easier.

A personnel system is a good example. Here, government regulations relating to company personnel policies have been changing continually in many countries. New regulations have required companies to offer additional health and retirement benefits and options, new types of reporting on hiring and employee safety practices, and so on. A personnel system must be changed quite often.

Data processing departments may worry that DMS applications will not make efficient use of computer resources. If this is really a problem, the most crucial modules can be tuned; DMS experts tell us they can often cut processing time appreciably by just re-designing the most heavily used portions. If the DMS routines still use more resources than desired, another way to increase efficiency is to re-code certain portions in COBOL or assembler. A good number of the DMS allow subroutines written in other languages to be called upon by the DMS applications.

So, for on-line, interactive applications particularly, some of the prototype system can become part of the final system.

How prototype development differs

The application development cycle using prototyping does differ from the conventional process. It differs in three ways—in the tools used, in the skills needed, and in the procedures followed.

Different tools

Prototyping requires software tools that allow designers or programmers to create a working system in a very short time. We found the fol-

lowing types of tools in use: (1) DMS with an accompanying non-procedural language, (2) application development systems with an accompanying procedural language, (3) application generators, and (4) libraries of re-usable code.

Data management systems (DMS). As we described in the May 1981 report, a DMS allows the user to specify *what* needs to be done rather than *how* the work must be performed. A non-procedural language provides this user interface. Additionally, a DMS can have many useful functions already coded; the user need only specify the parameters of the operation and the files to be used. So numerous functions can be performed quickly and easily to set up a working prototype. These functions include allocating storage space for the data, creating data definitions, select and sort, retrieving answers to queries, and report generation. Some DMS also provide facilities for defining formats for data entry on a CRT screen. The designer need only enter the appropriate titles in the location where they should appear on the screen, and then indicate data entry field locations, field lengths, and field validation criteria.

DMS also have either DBMS or file management facilities so that updating the files is done easily, usually without writing any code. The DBMS may also take care of the security and recovery aspects of the database. So the programmer is left with more time to concentrate on the design of the prototype, rather than coding routine functions.

Products we would include in this area are RAMIS, from Mathematica Products Group; FOCUS, from Information Builders; NOMAD, from National CSS; INFO, from Henco Inc.; and many others. For a listing of these and other DMS products, see Reference 6.

Application development systems. The term 'application development system' can be confusing because it is used to refer to different types of tools. We think of application development systems as tools designed for programmers—tools that would rarely be employed by typical end users because they require a procedural language, such as COBOL. A DMS, on the other hand, is designed so that it *can* be used by end users. However, a DMS can also be, and often is, used by programmers as well.

Application development systems can provide many of the same facilities as the DMS. As we say, though, use of a procedural language is generally required. Products we would include here are Automated Development Facility (ADF) from IBM, ACT/1 from Art Benjamin and Associates, and PRIDE/ASDM from M. Bryce and Associates, Inc. Again, see Reference 6.

Application generators. Program generators have been around for some years; in fact, numerous user companies have written their own. Also, we discussed the use of an early version of the GENASYS application generator in our September 1975 issue; an application generator has a broader scope than a program generator. Many of the vendors have enhanced their products to include: a data dictionary, on-line interactive use, screen format generator, and other DMS-like facilities for developing on-line applications. Generators were originally designed for data processing professionals, but some have been enhanced to the point where end users can now use them.

These generators produce procedural language source code; unlike the DMS, the resulting application does not run under them. Using the David R. Black Generator, for example, the programmer selects from a menu what type of module he wishes to create. The system then initiates a question-and-answer dialog, in which the programmer supplies the variable information. When this form of 'coding' is complete, the generator produces the actual source code. It can generate 100% of the code for standard file entry and update programs, including formatted input screens and output reports. For more specialized application logic, source code can be hand-coded and added to the generator's code. So generators can also provide a means for creating a prototype quickly.

Libraries of re-usable code. One of the points that stood out in our research for the October and November 1979 reports on programmer work-stations was that these products encourage the use of re-usable code. Modules can be quickly retrieved from storage, changed if neces-

sary, and then used in a new program. By creating an on-line library of such modules, programmers can more rapidly create a working system. Jenkins pointed out to us that such libraries are also being used for prototyping purposes.

Different skills

The prototyping environment does require different skills for the data processing professional. Jenkins and Naumann note that, with prototyping, interviewing skills are not as important as they are with conventional methods. This is because user specifications are no longer based on how well the analyst interprets the users' spoken requirements; instead, the specifications are based on the demonstrated, working prototype. Further, the analyst does not need to uncover all requirements at the beginning; each version of the prototype helps the user to successively refine his requirements and to identify missing ones.

Prototyping also requires the data processing professional to spend more time with users and less time coding—hence the professional must be more people-oriented. Programmers who prefer the 'art' of programming might not particularly enjoy this new environment. There is still some coding to be done, but far less than in conventional development, especially if part of the prototype is used in the production system.

On the users' side, Jenkins and Naumann point out that prototyping requires the most knowledgeable users—generally the managers—to work on the system design, because they must be able to define the entire problem and choose among alternative solutions. One former data processing manager, now a consultant, pointed out to us that, on projects at his former employer, prototyping has required a lot more of these users' time than had been expected. Such time commitments need to be considered at the outset.

Different procedures

Prototyping definitely requires different procedures from the conventional development cycle. Peters (Reference 5) points out that this difference is fundamental. In the conventional cycle, user requirements are defined in a specification document, which is approved and signed by the user. Only after that phase is completed does the design of the system begin. In prototyping, Peters sees requirements and design evolving together. The development of the prototype deals with both—the requirements from the users' view and the design from the designers' view. Even when the prototype is used to create written specifications, design aspects are considered as well.

Jenkins and Naumann identify four steps in the prototype development process which precede the development of a production version of a system. We found that these steps depict quite well what we learned at the user companies we visited.

Step 1: Identify users' basic requirements. The first step involves uncovering only the users' most basic and evident requirements. The size of the project determines the amount of time spent on this step. For smaller projects, the designer generally talks with a few users for a short time to find out what the problem is and what they expect the system to do. Together they may define the information to appear on certain reports and data input screens. The designer might also want to get some statistics about the expected volumes of the various types of transactions the system will process. This initial requirements study can be informal and need not involve any written specifications. For larger systems, a design *team* may need to spend a few weeks preparing a first-effort requirements document, in order to get a better idea of the numerous functions the system will need to perform.

Step 2: Develop a working prototype. The important point in prototyping is that the designer takes the notes developed in the user discussions and *quickly* creates a working system. For example, when using a DMS, the designer can use the default values in the report generator to create standard report formats, rather than define the layouts of each report. Also, the prototype need only perform the most important, identified functions. The designer creates a relatively small file of data, enough so that users can experiment with the system. Within a few days, for a small

system, or a few weeks, for a larger system, the designer can develop the first prototype.

Step 3: Use the prototype. This step begins with the designer demonstrating how the prototype works to a small group of users. During the presentation, the users may request some changes. Some of these may be made right then—others can be reserved for later. If the system is not really what these users need, the designer may discard the prototype and start over again. If the system meets some of the users' requirements, and is usable, then the designer may teach them how to use the system so that they can experiment with it and evaluate it more fully. Generally this experimentation lasts for some specified time period, such as a couple of weeks. During this time the users make notes of all the changes they would like made.

Step 4: Refine prototype. Next the designer and users discuss the desired changes, and decide which ones should be included in the next version of the prototype. The designer then creates that version—either by starting over again or by extending the current version. After the changes have been made, the cycle returns to Step 3—the designer demonstrates the new version, instructs the users on its operation, and lets them use it for awhile. Steps 3 and 4 are repeated until the system fully achieves the requirements of this small group of users.

At this point, two alternatives are available. One is for the designer to introduce the prototype to a much larger group of users for their experimental use, to see if additional requirements come to light. The other alternative is, if enough users are satisfied with the prototype, it can be demonstrated to management to gain approval for the production version.

The point was made to us that the best way to demonstrate the prototype's usefulness to management is to have the managers bring to the demonstration questions they would like the system to be able to answer. But rather than have them operate the system themselves, the designer should enter their queries.

One possibility at this point, of course, is to continue to use the prototype as the production system. This is often a viable alternative, particularly on small projects which do not have a high volume of transactions. Unless the economics of running the prototype are too severe, we suspect that many companies will chose not to re-code it. For one thing, maintenance of the prototype probably will be much easier and less costly than maintenance of a re-coded production system.

However, if a production version is desired, after the approval for it has been obtained, the designers have three options. If the prototype is created by assembling modules of re-usable code, the final prototype may be close to the production version, with only some additional work needed to complete it.

Second, some parts of the prototype may be re-coded to operate more efficiently. We have seen this approach taken on a system that had batch updating programs; these programs were re-coded to increase efficiency. For medium size and larger prototypes developed using a DMS, this alternative seems to be the one most often chosen. The designers leave the portions that are most likely to change in the DMS, and they re-code the most heavily used—but at the same time, less dynamic—modules.

The third option is to re-code the entire system. If, for example, the DMS is available on the development machine but is not available on the production machine, this option may be necessary. Or perhaps the system is to run under a DBMS that does not provide many DMS facilities.

One case we are familiar with, where an entire prototype was re-coded, was where the end users created the prototype in order to present their requirements concisely. The prototype did not represent an entire system, however. The data processing department then re-coded this prototype and embedded it in a production system.

How many resources does prototyping use? From our talks with companies, the prototyping of end user requirements appears to take about 10% to 20% of the development time for large systems (the developers of large systems being the only ones we found who kept track of this kind of information). These users felt that prototyping did not shorten this phase of the development cycle. However, it *can* shorten the remain-

der of the development cycle, and it definitely *does* ease maintenance. In support of this last point, the RCA personnel system discussed above is maintained by just one person, because it was well designed through prototyping and because the most dynamic portions have been left in the DMS form.

Of course, if the final prototype becomes the production version, then development time *is* shorter—and maybe dramatically shorter—as compared with conventional development methods.

When are prototypes appropriate?

A number of authors believe that prototyping should be tried for all systems, because the approach presents a better and earlier view of the worthiness of the project and its design. It encourages heavy user involvement in the requirements and design phases, and in so doing, it accommodates change. McCracken feels this is the major benefit of prototyping—systems must be expected to change because automation of any system leads to change. Prototyping takes this into account. It is especially useful where user requirements can not be clearly defined until after much study, or where the needs are expected to change quite often.

Further, numerous authorities believe that prototyping is ideal for developing decision support systems, because users of these systems—typically managers and their immediate staff members—cannot foresee how they will use the systems.

With growing experience in prototyping and an increasing number of tools to aid designers create software prototypes, the time appears ripe to consider this approach to application system development.

REFERENCES

1. Berger, Paul and Franz Edelman, "IRIS: A transactions-based DSS for human resources management," *Data Base*, A Quarterly Newsletter of ACM's Special Interest Group on Business Data Processing, Association for Computing Machinery (1133 Avenue of the Americas, New York, New York 10036), Winter 1977, pp. 22-29; price $3.50.
2. Gomaa, Hassan and Douglas Scott, "An APL prototype of a management and control system for a semiconductor fabrication facility," *Proceedings of 1980 APL Users Meeting*, October 6-8, 1980, Toronto, Canada, I.P. Sharp Associates Ltd. (156 Front St. West, Toronto, Canada), pp. 73-83; price $15.
3. Jenkins, A. Milton and J. David Naumann, "The prototype model as a MIS design technique," Discussion Paper No. 163, Graduate School of Business, Indiana University (Bloomington Indiana 47405), September 1980, 30 pages; price: free.
4. McCracken, D. D., "Software in the 80s: Perils and promises," *Computerworld Extra!* (375 Cochituate Road, Framingham, Mass. 01701), Sept. 17, 1980, pp. 5-10; price $1.25.
5. Peters, Lawrence, "Users requirements and software specifications," Session No. 27 at 1980 National Computer Conference, May 19-22, 1980, Anaheim, California. A cassette tape of this session can be obtained from On-the-Spot Duplicators, Inc., 7224 Valjean Avenue, Van Nuys, California 91406; price: $8.00 including shipping and handling. No paper of the session is included in the published proceedings.
6. For a free listing of tools useful for prototyping, including the ones mentioned in this issue, write to EDP ANALYZER, 925 Anza Ave., Vista, California 92083.

Prepared by:
Barbara C. McNurlin
Associate Editor

COMMENTARY

PROBLEMS WITH PROTOTYPING

A consultant friend of ours, during a discussion about this issue, asked, "What are the problems with prototyping? How can data processing management control its use and keep it within bounds?" These are fair questions, so we thought that we would devote this Commentary to answering them.

One problem with prototyping was reported to us by Mattel Toys, as discussed in our July 1977 issue. In this instance, prototyping was used to quickly give a manager a new system he requested. He used the system for awhile, but then his use slowed and finally stopped. The systems department, in analyzing this experience, came to the conclusion that the system only solved a symptom and not the basic problem that the manager had faced. "It cured the itch, not the disease," they told us.

So one possible problem with prototyping is that it may encourage the glossing over of the system analysis portion of a project. Hence the basic problem, for which the system is desired, may not be identified.

Jenkins and Naumann's four-step approach to prototyping, discussed earlier, begins by identifying the user's basic requirements. So this step need not be glossed over.

In our July 1977 issue, we discussed one problem with prototyping put forward by big-system developers. It often is not clear just how a big system can be divided, in order to build a prototype one part at a time, until a thorough requirements study has been made. It is hard to see at the outset how the multiple parts will impact one another.

A subscriber in a multi-location organization wrote us about a problem he faces in the use of prototyping. People in his systems department resist the use of prototyping for large application systems that serve multiple user sites, on the grounds that it would be too expensive to have users at each site operate the prototype. The argument has validity, but it seems to us that the 'conventional' approach to this problem would be one practical solution for many such organizations. With this conventional approach, management appoints a group of users to represent all users. It is then up to the group to see that they adequately represent the others.

In addition, two of the user experiences discussed in this issue relate to this point. The RCA personnel system was converted one division at a time; changes could be made as the need for them became apparent. Also, the G.E. prototype was available over a time-sharing network, thus allowing users at various sites to test it.

Joseph L. Podolsky, Division Controller of the Hewlett Packard Microwave Semiconductor Division in San Jose, California, wrote an article which appeared in the November 1977 issue of *Datamation* ("Horace builds a cycle"). In the article, he describes both the advantages and possible problems

with prototyping (which he called 'recursive development'), both from the users' viewpoint and from the systems department's viewpoint. We contacted him to learn his current views on these problems.

In practice, he told us, the users' problems have proved to be minimal. Users enjoy their role in the use, testing, and refining of the new application systems. They are grateful for the support they get from even a less-than-perfect prototype.

One danger, though, is that users can become so happy with the prototype that they want the systems people to start working on something else, rather than cleaning up what is, in the minds of the designers, an 'early version' of the prototype.

The systems department's problems with prototyping are more serious, he said. One problem involves the acceptance of the method by the systems people. Prototyping is so different from the conventional, accepted method of system development that its use is often resisted. The skeptics take some convincing. One way to do this, he says, is to point out to the skeptics that prototyping is, in effect, an extension of the modelling process used in conventional design methods—by block diagrams, flow charts, and so on.

In his article, Podolsky listed five concerns about the effect of prototyping on the systems department. Very briefly, these were: difficulty in resource planning, difficulty in making a good decision on whether to enhance an old system or build a new version, the boredom that the Nth iteration of a system may bring to the developers, and problems associated with keeping the systems staff abreast of each system and of testing one iteration after another. These concerns have, in fact, proved to be valid ones, he says.

In addition, prototyping seems to bring about a reduction in the discipline that is needed for proper documentation and testing of a new system, he said. The system is so easily changed that keeping documentation up to date is a problem; as is often the case, it is treated too casually. Also, the systems department tends to assume that the users will do the testing of the new system. It is expected that any gaps or oversights in this testing by the users can be found and corrected 'later.' So testing may not be as thorough as desired.

Even with these concerns, Podolsky is an enthusaistic supporter of prototyping. The joint user-designer involvement allows for incredibly productive working relationships, he says. In fact, he believes that as users learn more about it, prototyping may well become inevitable. So it behooves all data processing executives to learn to use this powerful tool creatively and to manage it effectively, he says.

These, then, are the problems with prototyping that we have come across. None seem particularly troublesome. The advantages of prototyping, as discussed in this issue, would appear to greatly outweigh the problems.

SUBJECTS COVERED BY EDP ANALYZER IN PRIOR YEARS

1978 (Volume 16)

Number	*Coverage*
1. Installing a Data Dictionary	G
2. Progess in Software Engineering: Part 1	H
3. Progress in Software Engineering: Part 2	H
4. The Debate on Trans-border Data Flows	L
5. Planning for DBMS Conversions	G
6. "Personal" Computers in Business	B
7. Planning to Use Public Packet Networks	F
8. The Challenges of Distributed Systems	E,B
9. The Automated Office: Part 1	A
10. The Automated Office: Part 2	A,D
11. Get Ready for Major Changes	K
12. Data Encryption: Is It for You?	L

1979 (Volume 17)

Number	*Coverage*
1. The Analysis of User Needs	H
2. The Production of Better Software	H
3. Program Design Techniques	H
4. How to Prepare for the Coming Changes	K
5. Computer Support for Managers	C,A,D
6. What Information Do Managers Need?	C
7. The Security of Managers' Information	C,A,L
8. Tools for Building an EIS	C
9. How to Use Advanced Technology	K,B,D
10. Programming Work-Stations	H,B
11. Stand-alone Programming Work-Stations	H,B
12. Progress Toward System Integrity	L,H

1980 (Volume 18)

Number	*Coverage*
1. Managing the Computer Workload	I
2. How Companies are Preparing for Change	K
3. Introducing Advanced Technology	K
4. Risk Assessment for Distributed Systems	L,E,A
5. An Update on Corporate EFT	M
6. In Your Future: Local Computer Networks	F,B
7. Quantitative Methods for Capacity Planning	I
8. Finding Qualified EDP Personnel	J
9. Various Paths to Electronic Mail	D
10. Tools for Building Distributed Systems	E,B,F
11. Educating Executives on New Technology	K
12. Get Ready for Managerial Work-Stations	A,B

1981 (Volume 19)

Number	*Coverage*
1. The Coming Impact of New Technology	K
2. Energy Management Systems	M
3. DBMS for Mini-Computers	G,B
4. The Challenge of "Increased Productivity"	J,K
5. "Programming" by End Users	H,G,B,C
6. Supporting End User Programming	H,G,B,K
7. A New View of Data Dictionaries	G,B,
8. Easing the Software Maintenance Burden	H,B,G,
9. Developing Systems by Prototyping	G,B,H

Coverage code:

A Office automation
B Using minis & micros
C Managerial uses of computers
D Computer message systems
E Distributed systems
F Data communications
G Data management and database
H Analysis, design, programming
I Computer operations
J Personnel
K Introducing new technology
L Security, privacy, integrity
M New application areas

(List of subjects prior to 1978 sent upon request)

Prices: For a one-year subscription, the U.S. price is $60. For Canada and Mexico, the price is $60 *in U.S. dollars,* for surface delivery, and $67 for air mail delivery. For all other countries, the price is $72, including AIR MAIL delivery.

Back issue prices: $7 per copy for the U.S., Canada, and Mexico; $8 per copy for all other countries. Back issues are sent via AIR MAIL. Because of the continuing demand, most back issues are available.

Reduced prices are in effect for multiple copy subscriptions, multiple year subscriptions, and for larger quantities of a back issue. Write for details.

Please include payment with order. For payments from outside the U.S., in order to obtain the above prices, *use only an international money order or pay in U.S. dollars drawn on a bank in the U.S.* For checks drawn on banks outside of the U.S., please use the current rate of exchange and add $5 for bank charges.

Editorial: Richard G. Canning, Editor and Publisher; Barbara McNurlin, Associate Editor. While the contents of this report are based on the best information available to us, we cannot guarantee them.

Missing Issues: Please report the non-receipt of an issue within one month of normal receiving date; missing issues requested after this time will be supplied at the regular back-issue price.

Address: Canning Publications, Inc., 925 Anza Avenue, Vista, California 92083. Phone: (714) 724-3233, 724-5900.

Microfilm: EDP Analyzer is available in microform, from University Microfilms International, Dept. P.R., (1) 300 North Zeeb Road, Ann Arbor, Mich. 48106, or (2) 30-32 Mortimer Street, London WIN 7RA, U.K.