020
I 2901 LSX.
no. 177
cop. 2

University of Illinois
Graduate School of Library and Information Science

OCCASIONAL PAPERS

ISSN 0276 1769

Number 177
March 1987

Prototyping a Microcomputer-Based Online Library Catalog

by

Susan S. Lazinger
and
Peretz Shoval

Prototyping a Microcomputer-Based Online Library Catalog

by

Susan S. Lazinger
and
Peretz Shoval

Contents

INTRODUCTION: GOAL OF THE RESEARCH

In this work an attempt will be made to examine and evaluate the application of prototyping methodology in the design of a microcomputer-based online library catalog. The methodology for carrying out this research will involve a five-part examination of the problem on both the theoretical and applied levels, each of which will be covered in a separate section as follows:

1. A discussion of the standard life cycle in information systems development as defined in the literature of systems development and, particularly, of library systems development.
2. A definition of prototyping as perceived in some of the more recent literature of systems development.
3. Presentation of the two distinct types of prototyping in use in software engineering with accompanying discussion of how development according to each of these types differs from the standard development cycle.
4. Presentation of the model of prototyping roles and how it differs from the models of roles for standard development cycles.
5. Adaptation of the model of prototyping roles to the specific configuration in the presence of which prototyping is suggested as a design methodology for a microcomputer-based online library catalog.
6. Analysis of the use of prototyping methodology in the design of a microcomputer-based catalog in the Library of the Graduate Library School of Hebrew University—i.e., a case study of the application of the model.
7. An evaluation of prototyping as a viable systems-development methodology for libraries.

THE LIFE CYCLE APPROACH TO SYSTEMS DEVELOPMENT

The Stages of Systems Development

Systems development is a relatively new branch of technology. Dating back only 20 to 25 years, it has nevertheless undergone evolution. In the 1960s, software developers began attempting to design and implement increasingly complex information systems, a significant percentage of which were unsuccessful—i.e., they failed to satisfy the demands of the envisioned application. Anthony Wasserman, in his excellent 1980 article on information system design methodology, cites a number of reasons, which include:

> The developers didn't orient the system to the end users, who then rejected the system as being difficult to learn or difficult to use.
>
> There were no intermediate steps during the project at which project management and customer could review progress and determine the system status.
>
> The application demanded the use of highly advanced technology, which was not generally available.
>
> There was little or no effective communication between the user community and the development organization.[1]

Because of these problems, it was often impossible to distinguish the successful from the unsuccessful projects until very late in the project cycle which sometimes lead to disaster. Every piece of software for every information system was custom-made following no consistent pattern or cycle and benefiting little from previous experience.

In the late 1960s and early 1970s an attempt to structure the unruly methods of systems development came from the manufacturing concept set forth by the young field of software engineering—the "life cycle" concept. This concept of a "software life cycle" sought to combine all the different techniques of software production with appropriate management techniques into a consistent framework. Wasserman defines the various steps of the engineering process that serve to delineate the activities of software development as including, "but...not limited to, requirements analysis and definition, design, manufacture, quality assurance, and maintenance/enhancement."[2]

It has been said with some justification that there are as many definitions of systems analysis as there are people who have written about it. The term is sometimes used to cover the entire cycle of systems development and sometimes used to mean only the part that is called in the more recent literature "requirements analysis and specification." It is always divided into a number of stages, but the stages can vary from four to eight or more depending on how the analyst subdivides them. Table 1 compares the stages of system analysis as defined in two works of the midseventies—Burch and Strater's *Information Systems: Theory and Practice* and Chris Mader's *Information Systems: Technology, Economics, Applications.*

Wyllys, in one of the series of articles on system design that appeared in the *Annual Review of Information Science and Technology* throughout the seventies, mentions the problem of nonstandardized terminology in the literature of systems development:

> For example, ROSOVE [see references and additional references for all authors cited here] calls the phases: requirements, design, production, installation and operation; HAYES & BECKER include a preliminary

phase—feasibility—and follow it with specification, design, programming, test, implementation; HICE, ET AL., concentrating on data processing system development, call their phases: definition, preliminary design, detail design, program and human job development, testing, data conversion and system implementation, and system operation and maintenance.[5]

Wyllys then proceeds to add his own contribution to the welter of definitions of the phases of the system development cycle: "Here the phases will be called: analysis, design, production, implementation, and operation."[6] Since his five-step definition is nonetheless an attempt to standardize the vocabulary of systems development it seems an appropriate framework for describing the phases in the development life cycle in a bit more detail.

In the analysis phase, the system design team begins by defining the overall purpose of the existing system and of the changes desired—i.e., the requirements for the new system. In identifying system requirements, various

TABLE 1

THE STAGES OF SYSTEMS DEVELOPMENT ACCORDING TO BURCH AND STRATER AND MADER

Burch and Strater[3]	*Mader*[4]
1. Systems analysis:	1. Statement of systems objective(s);
1.1. Definition and formulation of the problem;	2. Creation of alternatives;
2. Systems design:	3. Systems analysis;
2.1. Developing alternative designs and solutions;	4. Systems design;
2.2. Building models which formalize the alternative designs/solutions;	5. Systems implementation: 5.1. Computerizing the designed system:
2.3. Determining the cost/effectiveness of the design alternatives;	5.1.1. Programming;
	5.1.2. Debugging;
2.4. Making recommendations;	5.1.3. Database preparation, if any;
3. Systems implementation:	5.2. Installing the system;
3.1. Implementation of the chosen alternative.	5.2.1. Documentation;
	5.2.1. Training; parallel with its predecessor systems;
	6. Systems evaluation.

constraints on the new system—e.g., cost, schedule, state-of-the-art (limitations of available hardware or software)—are considered. Careful and integrated analysis of existing conditions, requirements and constraints should eventually lead to a formal statement of what the new system should do. This is often called the requirements analysis document. When the system design team and the organization that will use the system concur on the system requirements, the analysis phase ends and the design phase begins.

The design phase follows the analysis phase although the two can overlap. In the second phase the system design team (or person if it is a one person design effort) considers how to carry out the functions required in the proposed system. If the analysis has been carried out thoroughly, the design of the system will logically follow from data collected in the observational and analytical processes. Alternative designs/solutions are considered, cost and effectiveness of the alternatives are determined, and recommendations are made.

In the production phase the chosen alternative is translated into reality. This phase usually includes the acquisition of the required computer hardware or software and, if necessary, the actual computer programming and testing of the programs.

As the components of the new system are programmed and tested the cycle begins to move into the implementation phase which includes conversion. This phase concludes when each subsystem or component of the system has been completed, tested, and found to be acceptable; when the system has been documented; and finally, when those who will operate the system have been trained. While some systems theorists end their definition of the systems-development cycle here, Wyllys adds what he calls the operational phase (sometimes called the evaluation phase) since it is in this phase that the evaluation of the system continues, often leading to further improvements.

In a rather early work on information systems development (1968), Donald Heany breaks the process down into eight numbered steps and then indicates precisely in which steps the user is usually involved:

> It is convenient to visualize system development as a process with eight discrete steps: (1) establishing or refining an information requirement, (2) developing gross system concepts, (3) obtaining approval to detail a particular gross system concept, (4) preparing detailed system specifications, (5) testing, (6) implementing, (7) documenting, and (8) evaluating the system and the process followed in putting the system in place.[7]

Users of the system's output are coparticipants in its development, particularly in steps 1,3,5,6, and 8. Heany's inclusion of the user in all but the physical design and programming of the system is progressive thinking for someone writing so early in the history of the field. He demonstrates even further that he is ahead of his time in his stress on the iterative nature of the process, a feature of systems development that was not easy to carry out given the state of computer technology in 1968: "These steps in system development often overlap. It is not always possible to finish one step before going on to the next. Furthermore, the process is iterative; designers are often forced to recycle."[8]

Approximately a decade later, the recognition of the need to include the user in system analysis and design, as well as a dissatisfaction with the standard tools of systems analysis (flowcharts, English narrative to describe the proposed system) gave rise to structured systems analysis. The stages in the development life cycle remain approximately the same in this updated version of systems analysis, but the techniques employed to do the analysis (and later the design) have evolved. Wasserman, in discussing the latest techniques—including Structured Systems Analysis (SSA)—also prefers to call the entire predesign phase by an updated name: "In this article, the term 'requirements analysis and definition' (RA&D) will be used to encompass the traditional notion of 'systems analysis' and the activity of producing a system specification."[9] He mentions in passing that a large number of techniques have been devised, developed, and used to assist in the requirements analysis and/or specification phases of the software life cycle and then goes on to describe in detail a few of these methods. One such method is SSA which is intended to be used in conjunction with Structured Systems Design (SSD). SSA incorporates four separate concepts or tools which Wasserman describes as follows:

> (i) Data Flow Diagrams show the flow of data between various processing units along with the processes to be carried out and the stores of data that are created, accessed, and modified throughout the system. A process is described in non-procedural terms within the box. Each box may be "explored" into another level of detail (or multiple levels) showing additional data flow, error conditions, and similar items.
>
> (ii) A Data Dictionary records information about all of the various data items and constraints upon them.
>
> (iii)Immediate Access Analysis is based upon a relational data base design to show the data base and the types of operations that will be performed.
>
> (iv) Program Design Language ("Structured English") or Decision Tables are used to show the process logic for each process box.[10]

Of even more interest than the new tools proposed by SSA are the reasons that are given for their development. Gane and Sarson, in the first chapter of their basic text on SSA, identify four limitations of the pre-SSA analytical tools: (1) There is no "model" in data processing unlike in the engineering sciences—i.e., there is no way of showing a vivid tangible model of the systems to users before the system is actually in operation. The pictorial tools—e.g., the Data Flow Diagrams—of SSA are intended to give the user a better "model" of the system than was previously possible. (2) English narrative is too vague and long-winded. (3) Flowcharts do more harm than good since they trap the analyst into a commitment to a physical implementation of the new system prematurely—e.g., whether the input will be on cards or through a CRT. (4) There is no systematic way of recording user preferences and trade-offs.[11] This awareness of the need to give the user what Gane and Sarson call a vivid tangible model of the system is thus one of the primary goals of this updated version of systems-development technique. The next step toward the production of a "vivid tangible model," as we shall see in the next section, is the technique of prototyping.

Library Systems Development

The stages of library systems development, as described in the literature of the profession, also vary in name and number but involve essentially the same phases as in development of other types of systems. As table 2 shows, Robinson breaks the development cycle into eight stages while Corbin and Fasana define a six-stage life cycle.

It is worth noting that Fasana emphasizes that his division into six stages is somewhat arbitrary and approximate and, more important, that while in theory the steps in a systems study should be done in sequence, "in fact, the process is iterative and overlapping."[15]

Since every author writing on library systems development, like every author writing on general systems development, defines the stages slightly differently, while still grasping the life cycle in essentially the same way—i.e., as "the systematic and logical analysis of a problem and the design of a system to correct any of the inefficiencies or errors which exist in the current operations"[16]—there is no point in describing further the various development phases as defined by each author. What is significant in this consistency in the literature of library systems development and different from the literature of general systems development is the emphasis on the need for the librarian to be trained in the role of the systems analyst or at least to have a basic understanding of the techniques of the system analyst. Author after author emphasizes that a library system designed exclusively

TABLE 2
THE STAGES OF LIBRARY SYSTEMS DEVELOPMENT
ACCORDING TO ROBINSON, CORBIN, AND FASANA

Robinson[12]	*Corbin*[13]	*Fasana*[14]
1. Determine feasibility;	1. Project planning and management;	1. Preliminary study;
2. Data gathering;	2. New systems requirements;	2. The descriptive phase;
3. Data analysis (Procedure performance analysis);	3. System evaluation and comparison;	3. The analysis phase;
		4. Design and development phase;
4. System design;	4. System design and programming;	5. The implementation phase;
5. Specify system;	5. Acquisition of hardware, software, and vendor assistance;	6. Evaluation and feedback.
6. Program specification;	6. System implementation.	
7. System testing;		
8. Implementation.		

by nonlibrarians will never answer the needs of the library it is meant to serve.

Fasana says that while the technical assistance required by libraries—with the onset of library automation—can be partially supplied by technicians, the coordination of these nonlibrary technical specialists can only be done properly by librarians because the system that will be developed will eventually have to be operated by librarians. He concludes: "The need, therefore, for librarians to gain experience and some level of expertise in systems analysis to guide the design and development of automated systems and then to operate them is becoming increasingly critical."[17] Chapman, one of the foremost writers on library systems analysis, states that the systems study must be done by the library organization and not for it if the study is to be successful. Since the staff must be familiar with the recommended operational structure and procedures in order to carry out such a study, he concludes that systems analysis and data processing courses must be included as a required part of library education. Writing in 1973, he notes that "many library schools are beginning to recognize the need for an introduction to systems study and data processing," in recognition of the fact that "library systems analysis, evaluation, and design will be ineffectual unless done by persons who are trained or formally educated in librarianship."[18]

Carter and Griffin go one step further and propose that the librarian should know not only systems analysis but also programming in order to facilitate effective library system development. Carter poses a rhetorical question as to where the main responsibility for library automation should fall and then answers it unequivocally: "Who should be the person(s) responsible for performing systems analysis in a library when a probable result is automation of one or more technical processing functions? Probably, the most frequent answer is that the analyst, or at least the project director, will be someone who is also a librarian with some training in programming."[19] Griffin also advocates the training and use of the librarian/programmer/analyst. He adds to the obvious reason that a librarian understands libraries the additional argument that if the analyst is also the user that the motivation for effective systems development is heightened:

> It is unfortunate but true that the best environment for library programming is within the library by programmer-analysts who have had training and appropriate experiences as librarians. They know why libraries do the peculiar things they do, and they have some interest in doing things better. If they are on the library staff, they are visible and face the prospect of living with satisfied or dissatisfied users—a good incentive to make the system work well.[20]

This involvement of the user in the design of the system is one of the cornerstones of prototyping methodology and will be discussed in detail in the following chapter on the prototyping approach to systems development. The librarian as systems analyst is a key component in the adaptation of prototyping to a design methodology that is viable for developing a microcomputer-based online library catalog, and will be dealt with in detail in section 3.

THE PROTOTYPING APPROACH TO SYSTEMS DEVELOPMENT

Prototyping: A Definition and Rationale

As discussed in the previous section, Wasserman's 1980 article detailing the history of information system design methodology cites the "custom-made" method of the sixties, the life cycle approach of the seventies, and the improved life cycle approach—Structured Systems Analysis and Structured Systems Design—that arose in the late seventies. Gane and Sarson, in their textbook on SSA, stated that one of the major reasons for the introduction of structured systems analysis was the need for what they called a "vivid tangible model," which they claimed the pictorial tools in their book would give.

It is this same Wasserman—who in 1980 described SSA as the latest technique in systems development—who shows up just two years later as the author of one of the Working Papers from the ACM SIGSOFT Rapid Prototyping Workshop held in Columbia, Maryland in April 1982 and published in a special issue of *ACM Sigsoft Software Engineering Notes* in December of the same year. His paper opens in an almost apologetic tone as though justifying his about-face from his position as a major proponent of the structured life cycle approach only two years earlier. While commending the "systematic approach to software design and development," he catalogs the shortcomings that have caused him to change his views during the two years since his previously cited article. Since Wasserman states clearly and succinctly the essential argument against the life cycle approach (and by inference for the prototyping approach) which begins nearly every article on prototyping, it is quoted here at some length:

> In general, efforts to follow a systematic approach to software development involve putting increased effort into requirements analysis, specification, and design, with the expectation of lower costs for testing and system evolution, since the implemented system will represent a better fit to the user's needs and will operate more reliably than would otherwise be the case.
>
> We found much to commend such a systematic approach to software design and development, but also found one significant shortcoming: there may be a long delay between the early stages of analysis and specification and the actual delivery of a tested, documented, reliable system.
>
> ...Two problems arise when trying to follow systematic methods:
>
> (1) the user's needs may change significantly during development
> (2) the user has no system in the interim.
>
> When the system finally becomes operational, it may do what was originally specified, but user experience with the system may show that what is needed is quite different.[21]

Another problem with the life cycle approach frequently mentioned in prototyping literature is that the user does not always know precisely what his needs are or may be in the future.[22] Conversely, the system designer may not be able to understand what the user really wants even if the user himself clearly understands: "When we attempt to build systems with novel capabilities, we often imperfectly understand the true user needs."[23]

The solution to each of these problems, according to the authors who stated them, is a relatively new approach to systems development, the basic principle of which—in Wasserman's words—is "let's build one of these and see how it works."[24] First of all, this methodology directs itself to the problems of changing user's needs and the lack of an interim system:

> We believe that each of these problems could be alleviated to some extent if the intermediate products of the systematic development process included preliminary versions of the system. Such preliminary versions would be helpful to both the user, who would obtain something tangible to use, and the developer, who could determine the feasibility and correctness of the earlier work, as well as modify the preliminary system based on user experience.[25]

The hands-on preliminary version or versions of the system, according to Taylor and Standish, would also solve the problems of the user's not being sure of what he needs and the builder's not understanding what the user *says* he needs:

> Exposure to working systems is often a helpful learning method. Looking at a system design on paper may not be as effective as direct exposure to the system behavior, since the user can often understand the latter without technical training, whereas it takes technical training to examine a design and to imagine what its behavioral implications are....
>
> There is a learning process involved in articulating the true user needs, which involves exposing the user to a working initial version of the system and seeing if he is satisfied.[26]

There is one last argument for producing a working version of the system which the user can interact with at an early stage of the system's development. This reason was mentioned in the literature of library systems analysis with regard for the need to involve the librarian in the design of a library system. It is, of course, user involvement. In other words, when the user interacts with the working system, he is forced to think about the system's functioning, needs, and shortcomings. Thus, in listing the reasons for building a prototype, Wasserman concludes that "it would give the user a more immediate sense of the proposed system and thereby encourage users to think more carefully about the needed and desirable characteristics of the system."[27]

It is interesting to note that the need to interact with a working version of the system, before a final commitment is made to implement it, is recognized in systems-development literature of more than a decade ago. Corey and Bellomy, in their 1973 article on determining requirements for an automated library system, describe a strategy called "the pilot test approach" which could be viewed as a precursor to the contemporary prototyping approach:

> The pilot test approach to implementation also allows the requirements study to be shortened with little risk. As before, a pilot test approach may

be used even though a model requirements analysis was conducted; in fact it often is. But there are situations where the pilot was seen as an alternative to a model requirements analysis. The new system is tried only for a subportion of the procedures that would be impacted by the complete operational system. For instance, the new system might be installed at a branch instead of the main library, or it may be tried on just monographs and not serials. The pilot is watched closely and after a preestablished period of time, the decision is made to convert fully to the system or scrap it. Pilot projects have the merit of offering empirical evidence of a system's suitability for meeting library requirements. Hard data are superior to the predictions of the most highly credible studies.[28]

As we shall see, there are several significant differences between this "pilot test approach" of a decade ago and contemporary prototyping. First, there is no mention of iterative developmental stages: the pilot system is implemented, tried out, and accepted or scrapped. This was unavoidable in the early seventies because computer technology had not yet reached a stage permitting interactive and incremental design improvement. Second, the pilot project is only sometimes seen as an alternative to detailed requirements analysis; essentially it is intended to be utilized at the implementation stage, somewhat later in the developmental cycle than in modern prototyping. However, it does recognize the fact that no paper analysis can prove a system's suitability or lack of suitability for meeting the requirements for which it was designed like a working model.

Four years later, writing in 1977, Patricia Zimmerman lists the need for a dynamic model of the system—i.e., a model which permits modifications—as one of the cardinal principles of systems design. Still limited by the available technology at the time she was writing, she concedes that economic considerations may preclude a truly dynamic model but recognizes that extensive iterations of a paper representation of the system are the essential minimum required to prevent user dissatisfaction later:

Cost factors may well prohibit a truly dynamic model, but some form of paper representation may be adequate. A representation can be as simple as flowcharts along with sample inputs and outputs and may be dynamic as a result of iterations of this documentation. The need for this step becomes clear the first time a user says, "I didn't think the report would look like that—it's not what I *had in mind*." It seems impossible to avoid this reaction, the real damage is when it first occurs after implementation.[29]

Thus, even before we offer a formal definition of the prototyping approach, it is clear that the concept involves two essential ideas: (1) design by means of a functioning, working model of a proposed system; and (2) iterative refinement of that design. Both of these concepts are reflected in the many variations of names given to the prototyping approach in systems-development literature dealing with this method. Read and Harmon call it "iterative requirements analysis,"[30] Podolsky speaks of a "recursive development cycle,"[31] Keen terms it "adaptive design,"[32] MacEwan advocates "iterative development accounting,"[33] McCoyd and Mitchell describe "system sketching,"[34] and Smoliar discusses the "transformational approach to software development."[35] All of them refer to an approach that is defined in Naumann and Jenkins's seminal article on prototyping as the use of a preliminary system "that captures *the essential features* of a later system....A prototype system, intentionally incomplete, is to be modified, expanded, supplemented, or supplanted."[36] It is this second half of the definition—a system which is to be modified, expanded, supplemented, or supplanted—which brings us to the next step in our definition of prototyping—an analysis and description of the two distinct types of prototyping.

The Two Types of Prototyping

If we revise Naumann and Jenkins's definition slightly so that it reads "a prototype system...is *either* to be modified, expanded, and supplemented, *or* supplanted," we arrive at a definition of the two distinct types of prototyping described in the literature although not always clearly distinguished one from the other. The prototype information system which is intended to be supplanted by a later, separate production system is close in concept to the prototype engineering system. Like a prototype engineering model, its purpose is primarily to revise and complete the specifications for a later, more sophisticated model by giving the user a working model to see and the designer a working model to test against the user's expectations and specifications. The second type of prototyping—which is intended to be modified, expanded, and supplemented—is the type that is of primary interest in this work, both because it is the method on which the prototyping model for library catalog systems is built and, on a theoretical level, because it is unique to information systems design as Maryam Alavi points out:

> This use of the prototype system (as a system that evolves into the production system rather than a system which is replaced by the production system) reveals a difference between an information system prototype and a hardware or engineering prototype. In engineering fields, a prototype is usually only a provisional system and is often discarded after experimentation and replaced by the operational system.[37]

Thus prototyping can be broken down into two-system, "throw-away" prototyping, and one-system, "add-on" prototyping. Hans Keus, in an important article defining and clarifying the precise differences between two-system and one-system prototyping,[38] assigns them the names Rapid Specification Prototyping (RSP) and Rapid Cyclic Prototyping (RCP) respectively, and henceforth in this work they shall be referred to by these names.

Rapid Specification Prototyping

In rapid specification prototyping we generally deal with throw-away prototypes which are models of specified parts of the functional system. The use of these RSPs enables intense user participation in a relatively short period of the total system-development process. The objective is to increase communication between the system designers and the users in order to revise, refine, and complete the specifications. In this type of prototyping, no attempt is made to incorporate into the prototype all of the features desired in the final system, only those which are suitable for testing in a working model:

> Because of the nature and the application mode of the RSP only a specially selected set of quality factors is applicable. In general quality factors like maintainability, efficiency, portability, documentation and completeness are of little or no importance. However, in order for the prototype to fulfill its function properly, the following set of quality factors is prerequisite: speed of realisation, modifiability, testability, communicativeness, accessibility.[39]

In this type of prototyping the tools and the programming language in which the prototype is developed often differ from those with which the production system is built. In addition, while RSP involves a significant deviation from standard life cycle systems development, it does not replace the entire life cycle but only a part of it.

In an early (1978) article on prototyping, which does not refer to the add-on type of prototyping at all (probably because this technologically more advanced concept had not yet appeared by this date), Michael J. Earl presents a model which he calls "Prototype methodology within the systems development cycle." It is in fact a model of RSP—as opposed to Rapid Cyclical Prototyping—within the systems-development cycle, but it nonetheless illustrates graphically the place of rapid specifications prototyping in the life cycle process.

Note that in this model the prototype is built and tested at the analysis and design stages, after feasibility studies are performed. The prototype is then run through all the remaining steps of the life cycle—programming and testing, conversion and installation, and monitor and review. If it is decided to produce a second or third prototype, these subsequent working models are respecified, redesigned, reprogrammed, and retested until it is

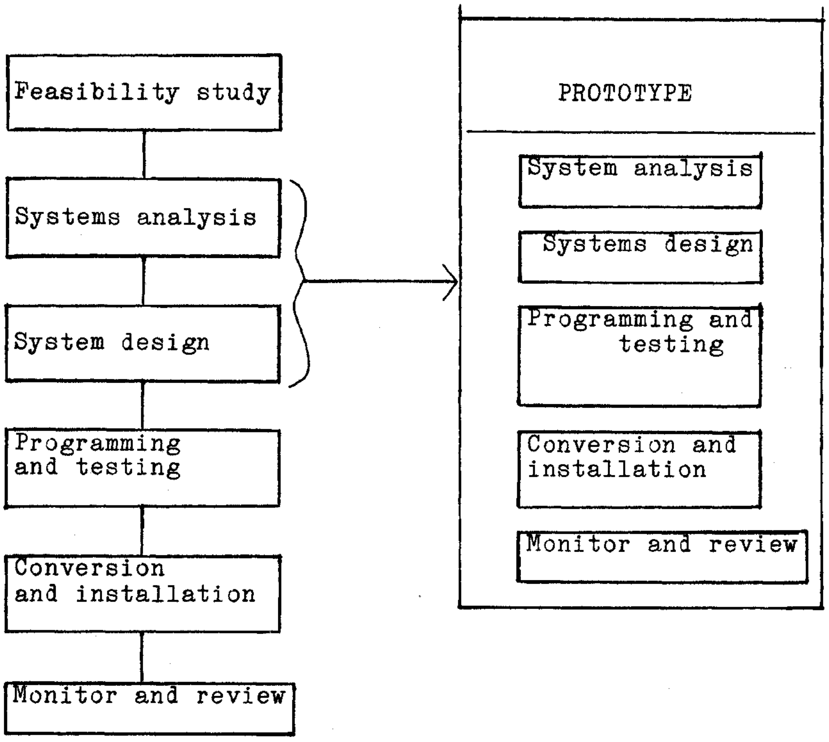

Figure 1. Rapid Specifications Prototyping (RSP) within the Systems Development Life Cycle (Based on Earl's Model)[40]

decided that the final system is ready to be built, ending the prototyping phase of development. The final or production model, which grows out of the changing specifications produced by testing each subsequent prototype, then proceeds through the remaining steps of the life cycle, as cited earlier. Thus RSP turns the life cycle into a loop of designing and testing progressive working models but does not replace it. Each prototype here is a distinct, separate model, replacing the previous model—i.e., the steps are *discrete*, not continuous.

Rapid Cyclical Prototyping

As we have seen, in the first of the two models for prototyping, a functional model is constructed to establish what the system should do—i.e., to refine rapidly identified basic requirements. This model is later discarded and replaced by a finished product, often implemented in a different (generally more traditional) programming language.

The second prototyping model is constructed on the assumption that the prototype will be refined and augmented to produce the final system. This model is based on the use of a very high-level programming language throughout (as opposed to the possibility of using a high-level language to program the prototype and a lower level language for the production system in RSP), a database management system, and/or an applications generator (the tools of RCP will be discussed at length in the next section). Naturally, the initial version, intervening versions, and final version are all in the same language for in effect they form a continuum, a single evolving system.

Rapid cyclical prototyping, the second model, affects the systems-development life cycle in a much more drastic way than rapid specifications prototyping. With this model it is not just portions of the life cycle that are changed. RCP *replaces* the traditional life cycle with a four-part cycle, the stages of which are defined with surprising consistency in the prototyping literature (unlike the stages of the traditional life cycle which are never defined exactly the same in any two works). In one of the few articles on prototyping that discusses the difference between the two types of prototyping (the majority of articles in the field do not relate to the difference between the two models), Bruce Blum points out the impact of RCP on the life cycle, although he doesn't call it by Keus's term RCP:

> The first model does not necessarily impact the traditional software life cycle. It may increase the cost and time of the initial stages to implement the prototype, but decrease the cost and time of the entire life cycle due to a better understood and more realizable target. However, the second model has a significant impact. It replaces portions of the middle of the life cycle with the process of prototype refinement. In order to quickly generate refinements, the model requires tools that provide rapid turnaround of prototypes. As a consequence, the end-product is more stable and closely aligned with the user's needs.[41]

Maryam Alavi, in a 1984 article on prototyping, presents a model (see fig. 2) showing the differences between the prototyping approach and the life cycle approach to information systems development. Although she does not specify that her model depicts specifically add-on prototyping or RCP, it is in fact a parallel model to Earl's earlier model—as Earl's model showed the place of RSP in the life cycle, Alavi's model shows the replacement of the life cycle by RCP.

This same four-stage cycle—identify, develop, use, and revise—is defined two years earlier by Naumann and Jenkins who, again without specifying

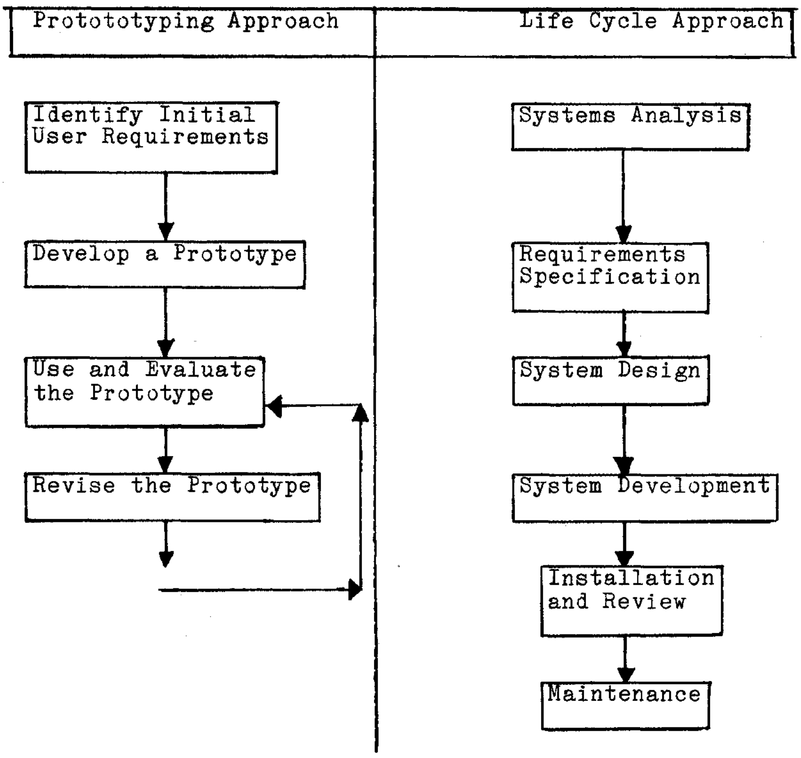

Figure 2. Rapid Cyclical Prototyping *v.* the Life Cycle Approach (Based on Alavi's Model) [42]

that their model describes add-on prototyping or RCP, present a prototyping model nearly identical to Alavi's (and possibly on which Alavi's later model is based). Their model, however, is constructed as a data-flow diagram rather than the flow-chart type structure of Alavi's (see fig. 3).

The first stage in this model, in which basic user requirements are identified, is left vague by both Naumann and Jenkins and Alavi. It must be assumed that the "identify" stage includes at least an abbreviated study and definition of systems objectives, as well as some sort of formulation of the system's general design and structure before the prototype is programmed, as illustrated in the case study in section 4. Alavi herself points out that the "quick and dirty" approach of providing the user with a working system

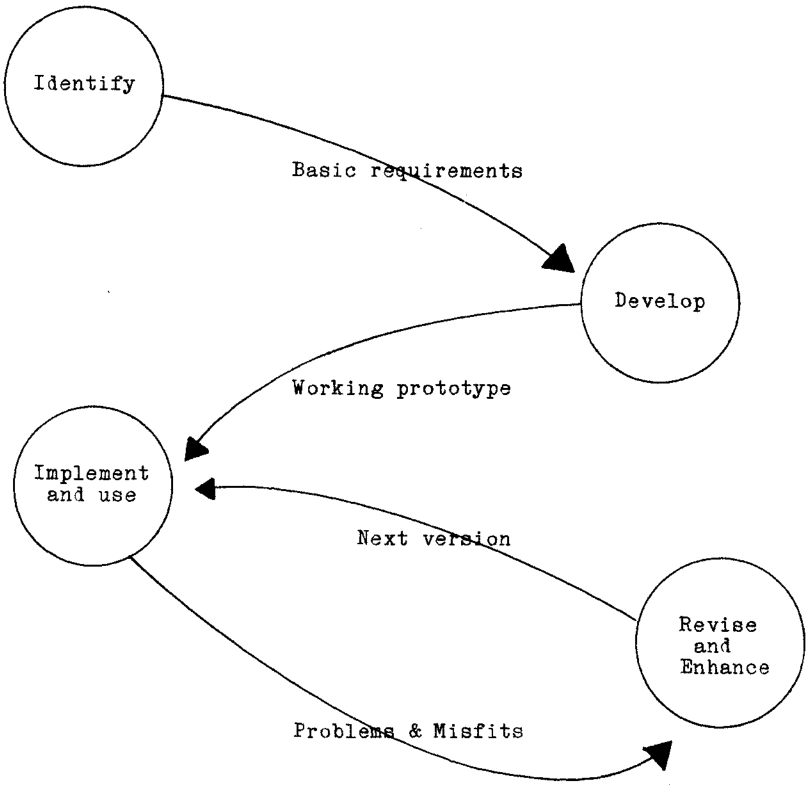

Figure 3. The Rapid Cyclical Prototyping Model (Based on Naumann and Jenkins's Model)[43]

before the goals and specifications have been thoroughly worked out can lead to problems such as poor planning, budgeting, and control of the project[44] at the same time that it helps solve other problems such as a system that does not meet the user's desires. The other key difference between this model and the RSP model, aside from its virtual replacement of the life cycle, is that this model is a *continuum* rather than a set of discrete prototypes. Once the system is programmed, the *same* system is modified online continually until it satisfies the user's requirements.

Prototyping Roles

The models for rapid cyclical prototyping of Alavi and of Naumann and Jenkins describe a four-step interactive process between two elements that Naumann and Jenkins call "user" and "builder." Both of these elements interact with the application system in a three-way iterative cycle. In this cycle, the initial version is defined, constructed, and used quickly. As problems are discovered, revisions are made to the working system in its user's environment. In order to define the working relationship between the user, the builder, and the system, Naumann and Jenkins present a second model which they call "Prototyping Roles." According to this model, the user is responsible for the functions of the application system starting with the definition of the system's basic requirements. By using and evaluating the system which the builder constructs according to these basic requirements, the user discovers problems and communicates them to the builder. The builder then constructs and revises the system in response to the feedback of the user as shown in figure 4.

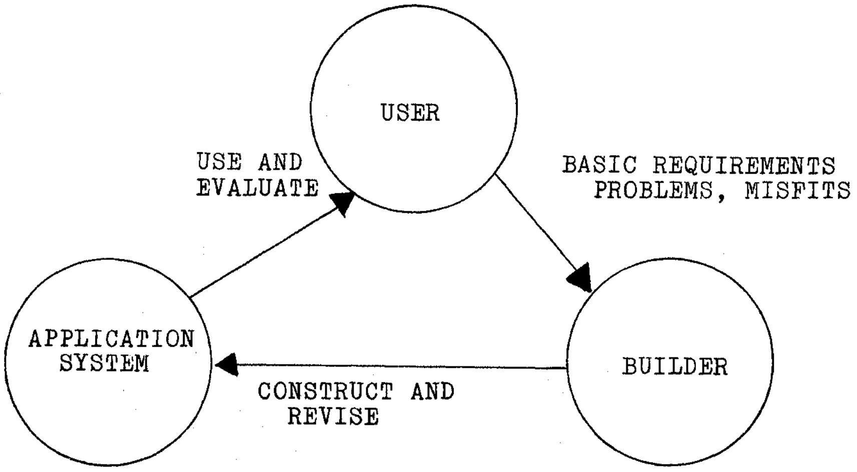

Figure 4. Prototyping Roles (Based on Naumann and Jenkins's Model)[45]

As we see from this model of prototyping roles, it calls for a radical departure from the roles of the user and the builder in the life cycle approach. Users play more active roles in prototyping than in the traditional systems-development methods. As Naumann and Jenkins point out, "in effect they are system designers who use and evaluate a system, and in the process, identify problems and suggest solutions."[46] It is the users who set the pace of development by the time they spend using and evaluating the prototype and who determine when the cycle of revision and refinement ends. The builder's role is to construct successive versions of the systems resolving conflicts between the user's needs and desires and the form of the system as constrained by technology and economics.

It is the model of prototyping roles that is of primary interest to us in the development of a microcomputer-based online library catalog. The adaptation of this model of prototyping roles to the specific configuration in the presence of which it is suitable for such an application is the subject of the next section.

PROTOTYPING ROLES AND LIBRARY SYSTEM DEVELOPMENT

The Model

Naumann and Jenkins's model of prototyping roles is eminently suitable for the in-house development of an online library catalog, notably when using a powerful microcomputer (probably a multiuser supermicro) with a hard disk. A computer with less capability than this is unlikely to meet the requirements of any but the smallest library. A microcomputer with a hard disk is suitable for in-house development of an online library catalog for a small- to medium-sized library (5,000 to 50,000 books), such as many special libraries and school libraries. Libraries larger than this are likely to require a minicomputer or main-frame, and will probably turn to the services of an outside organization to help them meet the complex requirements of a very large online library catalog. Thus the use of the model is recommended for libraries of a size that permits their collection to be stored on the disk of a microcomputer.

The software required for prototyping a library catalog includes a database management system with application generator capabilities. A precise definition of this type of software will be given later.

The personnel requirements for prototyping an online library catalog on a microcomputer are low: a librarian-analyst, with the type of background described in section 1—that is, a basic familiarity with computers and the principles and techniques of systems analysis—and an in-house programmer. Naturally there can be more than one librarian working on the system, and there can be more than one programmer. However, as the case study described in section 4 demonstrates, one of each, working side by side, can be very effective. In rare cases the librarian can also be the programmer, but for the most part the complexity of the programming required for an online catalog, even with sophisticated software, indicates the need for an experienced programmer. Unless the librarian is competent in programming beyond the level of even most computer-literate librarians, it probably will not be efficient to have the builder and the user manifested in the same person. In addition, even a librarian who is a competent programmer would do well to have input and feedback from at least one other person. Therefore, one builder and one user appear to be the minimum number of personnel recommended.

Thus Naumann and Jenkins's model of prototyping roles would be adapted for library system design to include a librarian/analyst (user), an in-house programmer (builder), and a microcomputer with a hard disk and an applications generator (system) as shown in figure 5.

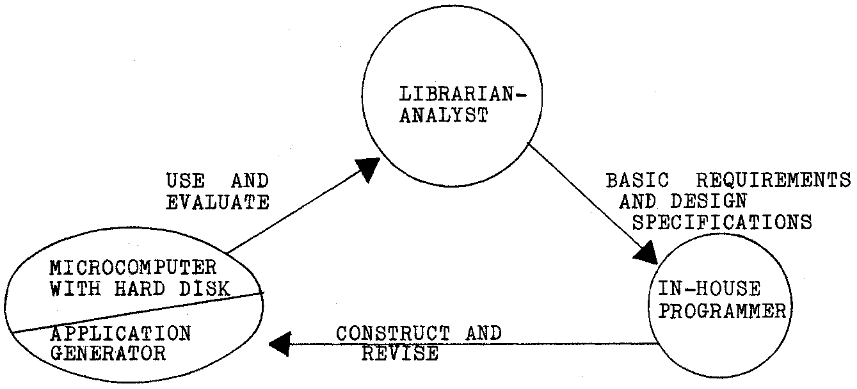

Figure 5. Prototyping Role Model for Library System Development

The Software

In view of the fact that the application generator is an integral part of the model of prototyping roles for online library catalog development, a discussion of the type of software required for the application of the model and of its significance for prototyping seems appropriate at this point.

Library software comes in three types: (1) off-the-shelf, ready-made systems (e.g., a fully-programmed catalog system, ready for input of a library's individual records); (2) a completely custom-developed system, programmed by a professional software house, usually in a traditional programming language; and (3) a customized system programmed with an applications generator. Our model utilizes the third method which can be compared to using a sewing machine to make one's own dress as opposed to buying it off the rack or having it made by a dressmaker. For a small- to medium-sized library with specialized demands it can mean the best of both worlds.

Since the first feature of an application generator is that it includes a database management system (DBMS) we will begin with a definition of a DBMS. Although databases—and thus DBMS applications—vary widely, there is nonetheless a more-or-less consistent set of DBMS features which can be identified. A DBMS is a layer of software between the physical database itself—i.e., the data as actually stored—and the users of the system. It is a set of programs that hides the physical organization (pointers, hashing, etc.) of the data from the user and provides a framework that allows the user to define and build databases without extensive programming knowledge. All requests from users for access to the database are handled by the DBMS, thus shielding database users from hardware-level detail in much the same way that programming-language systems for languages such as COBOL shield programming users from hardware-level detail.

The architecture of a database system is divided into three general levels—i.e., internal, conceptual and external. The internal level is the one concerned with the physical storage of the data; the external level is the one concerned with the way in which the data is viewed by individual users, and the conceptual, level defines a community user view. For example, the external level for users from the personnel department—i.e., the part of the database of interest to them—may be a collection of department record occurrences plus a collection of employee record occurrences, while users from the purchasing department may regard the database as a collection of supplier and part record occurrences. The conceptual model (sometimes referred to as the data model), on the other hand, is a representation of the entire information content of the database, a view of the total database content, or multiple occurrences of multiple types of conceptual records. For example it may consist of a collection of department record occurrences plus a collection of employee record occurrences plus a collection of part record occurrences plus a collection of supplier record occurrences. An external record (or logical record) is not necessarily the same as a stored record (or physical record). The user's data sublanguage—i.e., the subset of the programming language which is concerned with retrieval and storage of information in the database—operates in terms of external records, the records the user sees when he retrieves information. On the other hand, the conceptual record is not necessarily the same as either an external or an internal (stored) record. It is defined by means of the conceptual schema which is a definition of the union of all individual users' views. The database management system, as we mentioned, is the software that handles all access to the database. C.J. Date describes what occurs in a typical access request with regard to this three-part schema:

> Conceptually, what happens is the following: (1) a user issues an access request, using some particular data manipulation language; (2) the DBMS intercepts the request and interprets it; (3) the DBMS inspects, in turn, the external schema,....the conceptual schema...and the storage structure definition; and (4) the DBMS performs the necessary operations on the stored database. For example, consider what is involved in the retrieval of a particular external record occurrence. In general, fields will be required from several conceptual record occurrences. Each conceptual record occurrence, in turn may require several fields from several stored record occurrences...the DBMS must retrieve all required stored record occurrences, and then construct the required external record occurrence.[47]

In a DBMS, files, records, and keys are all defined by the user, and in most systems (including all applications generators) the user can design his own forms, tailoring them to suit the particular record type. Once the format of the record has been designed and entered, the DBMS enables the user to enter, change, or delete data by filling in the form he/she has created. The information may then be sorted and retrieved in various ways. The data is entered only once which reduces the chances of errors. Database management systems vary considerably in terms of programming skill requirements. Some of them were designed with programmers in mind for use in developing complex application programs—such as online catalogs. Others require no more programming skill than that needed for word processing.

The extent of the user-friendliness of the DBMS depends largely on its structure. The hierarchical model is based on a tree structure and groups items in a predefined order. It is limited as to the type of data which can be conveniently represented, since every element has a defined relationship with another more general or more specific type. The network approach differs from the hierarchical in that it does not limit the number of superiors or subordinates to which a record can be linked. However, because there is access from either direction, link patterns must be considered and the retrieval commands are complex. The third type of DBMS, the relational, is the last to appear on the scene chronologically and the most user friendly. It is based on the mathematical construction known as the relation, which can be represented graphically as a table with rows and columns. In this type of DBMS, data is arranged into tables of a fixed number of columns (fields) and a variable number of rows (records). This arrangement permits a high degree of data independence—since there are no formal internal link structures—and allows storage in a nonredundant form. The DBMS dBase II, is of this relational type, which is the recommended type for use with the prototyping model for the reasons noted earlier. Another advantage of the relational database is that it supports the use of fourth-generation languages (4GL) such as FOCUS, NOMAD, RAMIS, and INQUIRE. These languages are also known as *nonprocedural languages*, but Read and Harmon claim that this term is too limited because "in large, real world systems a liberal sprinkling of procedural code (e.g., IF statements) is essential for fine-tuning the main nonprocedural code."[48] Some of the advantages of these nonprocedural or fourth generation languages as Read and Harmon describe them are:

—90% reduction in physical code over third generation languages such as COBOL;
—performance of major rewrites simply and routinely when the organization changes;
—dynamic, relational database accessing, which lets the user collate data on-the-fly from all the key fields and data fields...more simply, any data can be retrieved from anywhere with no restrictions;
—language code comprising a wide range of powerful nonprocedural verbs with English syntax;
—interactive file editor, permitting on-line interactive update and retrieval;
—large new pool of programming talent (i.e., neither prior programming experience nor technical training are prerequisites for use of 4GL programming techniques).[49]

All of these features characterize dBase II, the software package used in the case study described in section 4, although the amount of previous programming experience required varies with the complexity of the application. Applications as complex as the online catalog of the case study require a programmer or someone with considerable programming experience, while simpler applications, including the first model, based on a straightforward use of dBase II's relational structure, can be programmed with minimal experience and little computer background.

The application generator, a type of software which is characteristic of, although not exclusively designed for, microcomputers, also varies in capabilities and degree of skill required for use. However, like the DBMS, the application generator has several consistent features. First it includes or "sits on" a DBMS. Second, it usually possesses (1) a screen generator, (2) a menu generator, (3) a user-friendly query language (a DBMS may have an interface with a host language rather than a query language of its own), and (4) a reports generator. Blum, in describing prototyping tools, defines the application generator as follows:

> An application generator is an interpretive system that is molded to a specific environment. A user of the system types in a specification of the application desired and the system responds by interpreting the specification and performing the desired function. Typical functions include database management and update, report generation, retrievals, graphics, statistical analysis, and screen layouts.[50]

Such high-level interpretive tools form the basis for the innovative type of system development offered by the prototyping role model. By offering a level of ease of modification not offered by traditional languages such as COBOL or FORTRAN, combined with relative ease of programming and formatting, they permit the kind of interactive, online system development which will now be described in the case study which forms the subject of the next section.

PROTOTYPING AN ONLINE CATALOG IN THE LIBRARY OF THE GRADUATE LIBRARY SCHOOL OF HEBREW UNIVERSITY: A CASE STUDY

The library of the Graduate School of Library and Archive Studies of Hebrew University is a small library in size—10,000 volumes and 115 journals. Nonetheless, the library required a bibliographic record for its online catalog which possessed features of a large library. Because it is the most extensive collection in library and information science in Israel, and the library of the oldest and only graduate library school in Israel, it was felt that it should set an example in its catalog. That is, full cataloging was established as the standard for its automated record, which was to contain all the information on the cataloging card plus Library of Congress subject headings for subject access to the material. This need for a long, complex record—coupled with the determination to develop an online catalog which permitted the type of subject access that a card catalog generally does not (i.e., by up to six or seven different subjects per volume)—indicated the need for a software package with capabilities beyond the requirements which the size of the collection might suggest.

The Library School library was in a unique position with regard to automation. While the rest of the Hebrew University Libraries were being automated through the university's specially designed ALEPH system, it was decided after much deliberation not to become a part of this network—at least for the near future. There were two reasons for this decision. (1) The library school possessed two multiuser microcomputers and an in-house programmer to run the computer laboratory for the students that these computers serviced. (2) Ironically, simultaneously, because the Library School is not part of a larger, wealthier faculty, it did not have the financial resources to connect to the ALEPH system and pay the monthly charge. A third reason was proposed, which in the final analysis did not figure in the planning of the system: that an in-house designed system could be used as a laboratory for the students. In the ultimate compromise between sophistication of design coupled with efficiency of storage structure and simplicity of programming (i.e., programming within the capability of library school students), it was decided to opt for the most highly sophisticated system possible within the limits of the equipment, the software, and the programmer's talents. This decision was reached at a relatively late stage in the planning only after it had been decided that full AACR2 cataloging was to be used in the online catalog's bibliographic record. The initial and primary goal was to design an online catalog that would facilitate retrieval of material by subject with the aid of Library of Congress subject headings, including material not covered in the manual catalog—so called "fugitive material": special reports and the various publications sent out periodically by library schools and other organizations. Subject retrieval in a manual (card) catalog required the typing of cards for each of the up to six or seven subject headings that could be assigned to a single book or report, an investment in employee time and catalog space that was considered unreasonable. Obviously, computerizing the bibliographic records so that all the data for each record would be entered only once would be an enormous saving in manpower and the physical space the six to eight cards per book would require in the card catalog.

As mentioned, the resources available for automation of the catalog were: (1) an 8-bit IMS microcomputer, with three processors and an 85 megabyte disk, which had been acquired for the computer laboratory used in teaching the students; (2) a librarian-analyst with approximately the background described in section 1; (3) an in-house programmer-lab technician; and (4) dBase II, one of the most popular and widely used of the commercial DBMS/application generator packages. Several other packages had also been acquired by the library school and were considered for use. They were either file management, rather than database management or, like Superfile, "free format" information retrieval packages. Since we wanted a record with defined fields, it was decided to program the catalog in dBase II. The programmer estimated that 12 to 15 megabytes would suffice to store the records for 10,000 books, so disk storage space was determined to be more than adequate. The programmer and librarian worked in close physical proximity so that the builder and user parameters of the model were present. The final parameter, the application generator dBase II, was judged suitable for the application because of its powerful capabilities and ease of programming. Since the software is such an important tool in the prototyping process, a detailed description of dBase II's structure and capabilities, as an example of the type of software required, would be appropriate at this point.

dBase II

Ashton-Tate's dBase II is probably the best known and most widely used of the growing number of database management systems designed for microcomputer applications. It is composed of a group of programs which allow the creation and manipulation of data files structured into the tables of rows and columns characteristic of relational database management systems. It utilizes a series of simple commands which allow the creation, editing, deletion, printing, and display of stored records.

The dBase II system differs from file management systems, such as Superfile (mentioned earlier), in that it allows the simultaneous processing of two data files and, in general, that it is based on the relational model. In addition, it possesses capabilities beyond those of a DBMS, capabilities which permit it to be used as an application generator as well:

> In addition to functioning as a data base management system with its own command language, *dBase II* can be viewed as an application generator that permits the creation of files of commands that operate much like programs. The creation of such command files is much simpler than conventional programming, however, and permits the implementation of complex applications much more quickly....
>
> In providing the capability to create and execute command files, *dBase II* goes beyond the capabilities of a data base management system and incorporates features associated with so called "application generators"—programs designed to facilitate the writing of other programs.[51]

In spite of the ease of creating command files, relative to the skill required to program in a conventional language, dBase II is nevertheless not intended for users with no previous computer experience whatsoever. For the most part, it is command-driven rather than menu-driven as are programs intended for end users with no programming knowledge. That is, the user is required to master a set of commands and not merely choose predefined alternatives from a list or menu displayed on a single screen. It is an intermediate step, a package for the intermediate user who has a certain amount of background and/or programming staff but whose resources do not extend to the hiring of expensive professional systems consultants and developers. In the article quoted earlier, Saffady relates to the requirements for the use of dBase II in a library environment: "While *dBase II* operations can be learned relatively easily and quickly, this is certainly not a software package for the naive user. Librarians acquiring *dBase II* for use in a particular application should budget sufficient time for the study of the user manual as part of the implementation process."[52]

Like other DBMS, dBase II files are composed of records. The maximum record length in dBase II is 1000 characters with a maximum of 32 fields. The maximum file size is 65,535 records, and the maximum length of any single field (fields are fixed-length) is 254 characters.

One of dBase II's unique features is that it does not use predefined keys. However, an index can be created simply by telling the system what the temporary key will be for sorting. Index files maintain pointers in ascending order to the contents of fields in the database files. By using database files with the appropriate index file it is possible to search the database file very quickly. Up to seven index files may be kept up to date when their database file is updated or edited or when records in a database file are deleted. Because the index files are maintained for these functions in the course of modifying the contents of the database file, it is not necessary to invert indexes as a separate process—the files are kept constantly up to date. The chief difference here from other programs is that the user does not have to specify any particular fields as keys while designing the data entry form. As a result he has the freedom to create various indexes based on any field at any time.

The dBase II software also allows the user to perform searches without specifying any index or key fields. Thus, for strategies that he knows will be used often, he can use the indexed method which is much faster. On the other hand, for doing unrestricted immediate searches, taking into account that they may take much longer, he has the option of an unindexed search on any field of his choosing at any moment. For example, a search on the "Imprint" field (which gives the place of publication) of a catalog system might not be required often enough to justify the disk space required to index it. Nevertheless, if a user desires occasionally to retrieve all books published in Chicago, for example, it can be done by searching the file sequentially. Saffady discusses this important feature also with regard to library applications:

> One of the most powerful and useful features of *dBase II* is the ability to combine the DISPLAY command with relational and logical expres-

sions in order to retrieve specific types of records. These capabilities are of considerable importance in library applications. Thus, for a data file consisting of bibliographic citations to magazine articles, the command DISPLAY ALL FOR SUBJECT = 'MICROCOMPUTERS' will display all records containing the value MICROCOMPUTERS in the SUBJECT field. The more complex command DISPLAY ALL FOR SUBJECT = 'MICROCOMPUTERS' AND JOURNAL = 'BUSINESS WEEK' uses the Boolean AND operator to select and display all records for articles about microcomputers published in *Business Week* magazine. *dBase II* also supports the Boolean OR and NOT operators, as well as the customary range of relational expressions. A substring operator can be used to display records containing a specified character string in specified fields.[53]

One final note on the capabilities of dBase II before we proceed to the actual application. By adapting the data structure appropriate to the system being developed, considerable flexibility can be gained. Thus, while the maximum physical record is 1000 characters, the maximum size for logical records depends only on the ingenuity of the system designer and the needs of the specific application as the next section will show.

The First Prototype: RSP

In designing the online catalog of the Graduate Library School of Hebrew University, both types of prototyping were used. In the first stage a model was programmed by the librarian, using relatively simple programming techniques, to determine the specifications and to see how dBase II's programs would function on an actual, if simplified, system. In the second stage, a much more complex system was designed with the specifications determined by the librarian and the programming techniques determined by the in-house programmer-technician. This system was then revised and refined online and interactively by entering a database of actual bibliographic records. Thus the first model demonstrates the use of Rapid Specifications Prototyping and the second model the use of Rapid Cyclical Prototyping in a library environment.

Once the goals of the system had been established (see references 47, 48, 49, 50), a thorough search of the relevant literature was begun on the subjects of microcomputers and libraries, microcomputer software, recommendations for designing original in-house systems and methods of retrospective conversion. In addition, in order to determine the sorts of requirements and capabilities it was reasonable to expect in a custom-designed online catalog, a thorough study of ALEPH was undertaken including field trips to observe the system in action in the newly-opened Hebrew University Library. Another trip was taken to the University of Haifa to view, and discuss with its designer, their online cataloging system HOBITS (Haifa Online Bibliographic Text System).

A preliminary set of specifications for field types and lengths, based on the original definition of goals and the adaptation of ALEPH's bibliographic record format to meet them, was then formulated by the librarian who began experimenting with programming in dBase II. While the library prototyping model proposed in section 3 uses only RCP—in the case of the library school—RSP was used to help determine the "basic requirements" of the prototyping model in a first system that was intended to be replaced. The librarian here used a "throw-away" model in order to get the "feel" of the DBMS, the general record structure which would be required, and its capabilities using normalized records and the simple relational programming techniques it offered. In programming the first model, the librarian did not relate to efficiency of storage or response time, problems which the limits of her programming experience and the time she would have to devote to mastering dBase II to the level of a professional programmer placed beyond her scope. The goal here was simply an online rather than a written preliminary model of record construction and retrieval possibilities—i.e., a specifications prototype.

The first librarian-programmed catalog system was based on a single file, composed of maximum-length, 1000-character records of the following structure:

FLD	NAME	TYPE	WIDTH	DEC
001	DDC	C	020	
002	INV:NO	C	020	
003	AUTHOR	C	100	
004	TITLE	C	125	
005	IMP:PP	C	100	
006	ADD:ENTR	C	130	
007	EDITION	C	050	
008	SERIES	C	125	
009	NOTES	C	125	
010	LC	C	020	
011	SUBJ:HEAD	C	125	
012	VOL:COP	C	050	
013	LOCATION	C	010	
	** TOTAL **		01001	

The 13 fields defined were: (1) the Dewey Decimal Classification number; (2) the Inventory Number assigned by the Jewish National Library to each book; (3) the first author or editor; (4) the title; (5) the imprint (place, publisher, date); (6) added authors, editors, compilers, or corporate bodies; (7) edition number and description; (8) series to which the book may belong; (9) notes on the book's form or content; (10) Library of Congress Classification number; (11) Library of Congress Subject Headings assigned to the book; (12) number of volumes and/or copies of the books the library possesses; and (13) the book's location—i.e., whether the book is on the shelves, with the material reserved for courses, or in the journals section. The added entries, series, and subject headings—all of which were fields that potentially included more than one entry—were input in continuous strings, and command files were programmed to allow Boolean searches on any part of these strings. Thus authors could be search utilizing an AND or OR operator which, in the case of authors, operated on any part of the combined fields of AUTHOR and ADD:ENTR. Series could be retrieved from any position within the series string, and subject headings could be retrieved with the AND or OR operators from any position within the string input into the SUBJ:HEAD field. The entire catalog was menu-driven with options for retrieval by author, title, series, subject heading, or key word(s) in the title.

A small database of 60 actual bibliographic records was entered into the catalog system to see if it worked, which it did. Thus it was established that a catalog, permitting Boolean searches on several fields and—of primary importance to the library school—subject searches, could be programmed in dBase II. The field lengths were determined to be adequate for most records but not for all thereby giving the librarian-analyst a fairly clear idea of the changes in field length and type which would be necessary for the next version of the catalog. This first model of the online catalog was never considered as a viable system for permanent use. It operated on unindexed files and therefore was unacceptably slow once more than 50 or 60 records were input. In addition, it was highly wasteful of disk space since it used fixed length fields long enough to accommodate most occurrences in most bibliographic records. Clearly a more efficient structure would have to be programmed. Just as clearly the first prototype had demonstrated that a sophisticated online catalog programmed in dBase II was feasible and that the field structure and lengths were approximately, although not totally, satisfactory. The next step was to take the specifications determined through the first prototype and turn them into a catalog system that was viable for entering 10,000 bibliographical records so that the slow response time and inefficient storage structure could be improved.

The Second Prototype: RCP

Phases 1 and 2: Identify and Develop

The online catalog designed by the programmer from the librarian's definition of "Basic Requirements and Design Specifications" took advantage of the ease of programming and modification offered by dBase II's high-level, nonprocedural programming language, but used a pointer system to store the data rather than normalized records. The entire system was designed to provide a flexible and inclusive external record while minimizing the amount of disk space required for the internal or stored records. Normalized records of the type used in the first model would require allocating to each stored record the number of characters per field that would cover all occurrences of the field in all bibliographic records. Say for example that the average author requires 30 characters, but there are also authors whose titles require up to 200 characters of space (e.g.: "Conference on the Future Role of Computerised Information Services at the University of London, London, 1977"). To allow 200 characters to the author field for each stored record would waste a prohibitive amount of disk space.

Another consideration was the need for certain features which the librarian wanted to copy from ALEPH and which one would normally expect only from a mainframe system. The most striking examples are the variable-length field and the repeating field. It was clear that fields long enough to accommodate all, or virtually all, entries in the AUTHOR/ADDED ENTRY, TITLE, SERIES, and SUBJECT fields could not be programmed as they were in the first model, as fixed-length fields containing the entire field entry. It was both wasteful of space and could not be accommodated within a record the maximum length of which was only 1000 characters. At the same time, these same fields were fields which often had more than one entry per bibliographic record—i.e., more than one author, series, or subject—and therefore there was also a need for repeating fields.

In designing the catalog, the working method agreed upon was that the librarian would indicate the specifications and the programmer would implement them through whatever programming techniques were feasible or inform the librarian that the specifications could not be implemented in dBase II. As it turned out, virtually all the desired features were able to be programmed into the system thereby demonstrating that the limits of dBase II's capabilities are indeed dependent to a great degree on the programmer's skill. Simple applications are available to beginners and quite complex applications are possible for advanced users.

The original record (i.e., the *user record*—the record displayed on the screen for input and retrieval) designed by the librarian and implemented by the programmer according to the model of prototyping roles was, like the first model, composed of 13 fields:

1-INVENTORY NO.
2-DDC NO. (Dewey Decimal Classification number)
3-LC CLASS (Library of Congress classification number)
4-AUTHOR ADDED/ENTRY
5-TITLE
6-EDITION
7-IMPRINT
8-COLLATION (pages, illustrations, size of book: not in previous model, but necessary for full cataloging)
9-SERIES
10-HOLDINGS (volumes and copies in library; replaced VOL:COP field)
11-NOTES
12-LOCATION
13-SUBJECT

The programmer solved the need for repeating fields and variable-length fields by programming a complex system that operates on the following principles. To create the effect of variable-length fields, he created a main file—CATALOG—(see the appendix for structures of all the files described later) with pointers to secondary files for authors, titles, subjects, or series entries over a certain length. Thus, for example, the main file contains a 40-character field for the first author (which is also the number of characters retrieved for this field in the short card retrieval option). Full information on authors is contained in the next field (AUTHOR:PTR) which is a 40-character field containing eight five-digit pointers to the full author records stored in a separate file, thereby allowing eight repeating fields. Each record in the AUTHORS file contains 40 characters for the author's name. Another one-character field in this same record, called EXTENDED, indicates by Y or N ("yes" or "no") whether it is necessary to go to the next overflow record for the rest of the entry on the author. A field called STATUS is included for indexing purposes so that only the first record of each author is found by searches. Thus an author with 40 characters in his name requires only the main record. An author with 80 characters in his name requires 2 records—the main record plus one overflow record. The last field in each overflow record—BOOK—is a five-digit field for storing the number of the book in the main file (i.e., each book input into the catalog receives a sequential book number which is its permanent number in the catalog). A similar structure is utilized for storing the other retrieval fields requiring variable-length repeating fields—i.e., title, series, and subject.

At the same time, the nonsearchable fields (e.g., edition, imprint, collation, holdings, notes) are stored in a file called EXTRAS. Each record in this file contains three fields: (1) WHAT, a one-character field which tells what field (or the user record) is stored in the EXTRAS record, (2) CONTENTS, which gives the contents of the field from the user record which it stores, and (3) BOOK, which here also indicates the book number assigned by the system. In actual use, the EXTRAS file was broken down into three files: EXTRASA and EXTRASB, to store odd number and even number books, respectively, in order to avoid exceeding the maximium number of records in a dBase file, and EXTRASX, a temporary storage file used for editing the fields which will eventually be stored in EXTRASA or EXTRASB (i.e., the program copies all of the information for a given book out of EXTRASA and EXTRASB into EXTRASX, does the editing, and then replaces it in EXTRASA and EXTRASB once it is clear how many records will be required for all the "extras").

The information stored in the main records—those contained in the file named CATALOG—is utilized for retrieving records in the short form (LC number, location, primary author, and title). This short form is displayed to the user doing an author or title search in the form of a list of hits that answer the search profile. The user then has the option of requesting the full form of a specific record or records that includes all the fields included in the user record described at the beginning of this section (note: several changes were made in this original record that will be discussed later).

In all, the online catalog system specified by the librarian and programmed by the in-house programmer was composed of 12 DBF or data files, 7 index files, and 41 command files. It allowed for input of up to 8 authors, 5 titles (the need for more than one title field was discovered in the course of prototyping tests on the system and will also be discussed late), 10 series, and 6 subject headings per book. It was programmed, once the basic specifications were established, within six weeks by a single programmer and then tested on a database of 400 actual bibliographic records.

Phases 3 and 4: Implement and Use, Revise and Enhance

The theoretical basis for prototyping is that even the most careful specifications analysis on paper cannot foresee every problem or demand on the system. Also, that only by *using* a system does one become aware of all the refinements necessary. In the case of the library school, preliminary specifications had even been based on a preliminary prototype. However, because of limitations on the programming skill of the librarian, the first model could not be modified greatly. This second system model was designed specifically to be used with Rapid Cyclical Prototyping—to implement, use, revise, and enhance online continuously until the user declared it was finished.

It was decided to enter a fairly sizable database (400 records) in order to cover essentially all variations in bibliographic records. In spite of, or perhaps because of strict cataloging rules, there are hundreds of minor variations in length and precise form of bibliographic records. Furthermore, variations in books and authors also introduce variations into catalog records. In the course of entering the 400 records, many changes were implemented, most of which could not have been anticipated without hands-on experience with the system.

Types of changes included, first of all, addition and subtraction of fields. The library was in the process of reclassification when the prototyping began, and a midstream decision to input only books that had been reclassified from the Dewey Decimal to the Library of Congress classification system eliminated the need for the field "DDC NO." Simultaneously it was found that there was a need for an alternate title field to provide for books that had titles in more than one language or which might be popularly known by less than the full title. As data entry continued, it was decided that there should be provision for up to four titles, so the ultimate structure allowed one TITLE field and three repeating ALTERNATE TITLE fields. The last field that was added in the course of testing was the RESPONSIBILITY field which describes the relationship of the author or authors to the book (editor, compiler, etc.). This field, while not adding any essential information not contained in other fields, is nonetheless a part of full cataloging and so it was decided in testing to add it. Thus the completed and revised user record, which comprised both the input form and the full card retrieval form, contained 14 fields of which fields 3, 5, 10, and 14 functioned as repeating fields for input as follows:

1 - INVENTORY NO.
2 - LC CLASS
3 - AUTHOR/ADDED ENTRY
4 - TITLE
5 - ALTERNATE TITLE
6 - RESPONSIBILITY
7 - EDITION
8 - IMPRINT
9 - COLLATION
10 - SERIES
11 - HOLDINGS
12 - NOTES

13 - LOCATION
14 - SUBJECT

The second type of revision implemented in testing was in the length or structure of the fields. These changes included:

1. establishment of how many repetitions of SUBJECT and SERIES fields to allow;
2. increase in the length of the NOTES field;
3. restructuring the SUBJECT field so that segments of subject headings were retrievable in any order;
4. creation of the author/added entry relator (i.e., name of author plus equal sign plus relationship, e.g., "SMITH, JOHN = ED.");
5. establishment of format for entering titles that begin with numbers or abbreviations (as a TITLE/ALTERNATE TITLE combination); and
6. determination of the spacing between segments of subject headings in the SUBJECT field (i.e., it was necessary here to distinguish between hyphens in words and dashes between segments).

The third and final type of revision implemented through testing of the prototype was the type of change that can be termed "human engineering"—i.e., aesthetic changes intended to make the system more user-friendly or attractive. These revisions in turn can be broken down into input and output modifications. Examples of input modifications are:

1. a decision to require use of the "return" key after all answers to all choices (to lessen the chance of mistakenly hitting a key and executing an order by accident);
2. rewording of instructions for response to the authority file verification when they were thought to be unclear;
3. addition of a program to implement corrections while still in the input mode (as opposed to a separate edit mode which requires exiting from the record being input);
4. further revision of this correction program so that it leaves the text being corrected in place for overtyping so that the inputter can see what needs correcting;
5. addition of a feature which constructs an inventory number for gifts to which the acquisitions department does not assign an official inventory number as it does to ordered books; and
6. revision of the correction program to display the full record before and after each correction.

Examples of output modifications are:

1. addition of alternating color modes (green and black stripes) for emphasis of main headings in short form;

2. decision to have user indicated whether he wants the full form displayed by a "yes/no" response rather than a "±" response which was felt to be confusing;
3. changing of the user instruction "list cards" (to receive a short form entries list answering the search profile) to "list records" which was considered more appropriate for an online catalog; and
4. addition of cross referencing for field entries that may have alternate forms.

While there may be some justification in claiming that modifications of the first type and perhaps even of the second type could have been rendered unnecessary by meticulous systems analysis (although it is debatable whether the investment in time to plan the system down to the last detail would have been worth it), it can be stated with some certainty that revisions of the third type—the finishing touches on the system's human engineering aspects—were changes that needed to be done online after the prototype had been developed. These were revisions of the type that come to light only with hands-on use of a functional system—in other words, only by prototyping.

PROTOTYPING AS A VIABLE SYSTEMS DEVELOPMENT METHODOLOGY FOR LIBRARIES: EVALUATION AND CONCLUSION

Advantages of Prototyping

The literature of prototyping is full of arguments for the use of this new and fashionable systems development method. Many of the advantages of prototyping have been mentioned, at least in passing, in the previous section. A summary of the advantages and limitations of prototyping as cited in the prototyping literature is nonetheless appropriate in conclusion.

Wasserman and Shewmake, in their statement of the four reasons for building a prototype, cover the most important advantages of prototyping in a single list:

> 1. it would enable the user to evaluate the interface in practice and to suggest changes to the interface;
> 2. it would enable the developer to evaluate user performance with the interface and to modify it so as to minimize user errors and improve user satisfaction;
> 3. it would facilitate experimentation with a number of alternate interfaces and modification of interfaces without having to redraw speci-

fication diagrams...; furthermore, it would be possible to generate the dialogue quickly and to make minor revisions in a matter of minutes;

4. it would give the user a more immediate sense of the proposed system and thereby encourage users to think more carefully about the needed and desirable characteristics of the system.[54]

Wasserman and Shewmake see this last reason as the most significant reason for prototyping, a judgment with which the present authors are inclined to agree. In nonprototyping development environments, there is typically a long time interval between the analysis of the problem and the construction of an operational system. By building a prototype and allowing the user to interact at an early stage in the system's development with a functional automated model, the user is forced to think more carefully about the task being automated and about the way in which operations will be requested. Two important results of this thinking should be (1) a more accurate understanding of the problem and improved communication between the user and the builder of the system, and, even more significant, (2) a final system that fits user needs far more closely than was previously possible.

Alavi, in a controlled study of experimental groups of "Users Exposed to Prototyping Approach" as opposed to "Users Exposed to Life Cycle Approach," reports that "mean scores indicate a higher utilization of the system designed by the prototyping approach."[55] Although these results cannot be considered conclusive, this finding may indicate that the prototyping approach does, indeed, tend to produce a final system that fits user needs more closely than a system produced by the life cycle approach.

Martin Feather suggests that developing a system by prototyping is an easier task than developing what he calls a "polished implementation":

> Our experience confirms our belief that prototyping is easier than developing a polished implementation, taking advantage of both the lack of necessity for optimal efficiency and freedoms not available to an actual implementation (e.g. use of perfect information).
>
> The methodology allows the incremental incorporation of techniques (gathered by adapting those of other researchers and by discovering new ones ourselves) and provides a convenient framework within which to introduce machine support for the various activities.[56]

McCoyd and Mitchell suggest that prototyping allows the discovery of defects in the system's specifications which even the most careful systems analysis might not discover. Their claim is that the user is not trained to conduct the analysis necessary to provide the builder with precise specifications. In addition, aside from this consideration (which is less significant if the user has some systems analysis background), the prototype provides an atmosphere of "excitement and discovery" that enhances user motivation to explore and improve the system and which no other development method provides:

> The value of rapid prototyping lies in the ability to see the system that has been specified. Despite the...simplicity of the systems we used to test...logical omissions in their specification were revealed by sketching them; corrections to the specification...were accomplished in minutes. While it can be argued that a careful, in-depth analysis of the requirements would have uncovered these deficiencies, the point is that a functional user is neither trained nor inclined to conduct this analysis. The prototype is not only easier to use, but since it creates an atmosphere of excitement and discovery, it increases the motivation of the user to explore all facets of his system description.[57]

Limitations of Prototyping and Conclusion

In spite of the wave of enthusiastic literature advocating prototyping (including this work) there are certain limitations on the method which should be kept in mind. First, prototyping is not appropriate to all systems in all environments. For example, it is not recommended for designing so-called "number-crunching" systems, or systems used primarily for mathematical calculations. Mason, et al., after stating that it is increasingly recognized that the traditional systems-development methodologies are not adequate for certain important classes of applications, go on to define these applications and describe the problems arising from the traditional methodologies:

> This is widely agreed to be the case for applications often characterized as Decision Support Systems (DSS), or Interactive Information Systems (IIS), and is often considered to be true for most business data-processing applications. The problems which lead to this conclusion include the apparent inability of development departments to deliver acceptable system function within acceptable time and cost limits. The ultimate users of application systems in the business environment are increasingly impatient with development departments which consistently deliver, too late, the wrong systems.[58]

Library systems fall into the second category—Interactive Information Systems—and therefore can be considered as a type of application that is, in fact, a candidate for the newer, prototyping method of development.

Blum and Houghton also temper their praise of the prototyping method with some caution, concluding that not every information management system application is appropriate for prototyping. Their reservations about prototyping involve its data dependence: prototyping can only be effectively carried out on a system into which data, in significant quantities, has been input:

> Rapid prototyping should provide an environment in which the final design can be verified before the system is installed. However, the data dependence of an IMS introduces several special problems. First, the use of the system relies upon the availability of a data base. System performance frequently cannot be evaluated with an empty data base, and the cost of data base creation may exceed the cost of prototyping. Next, a requirement for the heavy commitment of users' time to nonfunctional prototypes can doom an application before it can be completed....Consequently, it must be recognized that not every IMS application is suitable to rapid prototyping.[59]

In library systems, as the case study experiment showed, the availability of a database is not a problem. The bibliographic records of the card catalog can be entered easily—once the fields of the online catalog are established—even by relatively inexperienced users. Naturally it involves an investment in time, but the records input into the library school catalog were able to be utilized eventually for retrieval from the finished system. The changes which were carried out online on the prototype did not require throwing out records already entered. At most, relatively small modifications and additions were required.

Deletions were carried out automatically and globally by the system. The second factor—the very large investment in user time—is simply something that must be established before the working relationship between the builder and the user is set up. A user who is not sure that he will be willing and able to invest time in perfecting the prototype in order to arrive at a system that really answers his needs, should not enter into an agreement to develop a system by this method. Thus, prototyping requires both builders and users who are well-informed about the prototyping approach and committed to it and, in general, a supportive organizational environment.

Another limitation on prototyping is, of course, the technological one. A prerequisite to successful prototyping—in addition to committed users, an available database, and an application appropriate to the method—is the availability of technological tools that facilitate fast response to user requests. It is a method that cannot be implemented easily with traditional languages (COBOL, FORTRAN, etc.) because of the difficulty of making extensive changes in programs written in these languages. Thus an application generator (in the case of the proposed library model), or one of the high-level or nonprocedural languages written specifically with prototyping in mind (e.g., Ada, ACTS) are most helpful.

Even when all these limitations on the use of prototyping for systems development are taken into account and dealt with, there are still certain elements of uncertainty with regard to the success of systems developed this way. Taylor and Standish present a list of "what we do *not* easily learn from prototypes":

> (1) What it is like to live with the system for a while.
> (2) How easy a real system will be to alter.
> (3) How a system will behave when it is pushed to the extremes of performance (e.g., heavily loaded, various buffers nearly exhausted, displays saturated with data, etc.
> (4) How a system will interact with other elements in the software environment or related systems with which it should easily share data.[60]

In general, these problems relate to the stability of the system, how it will perform in the long run, and under conditions of everyday use. These are questions that no current method of systems development can answer, and in this respect prototyping, if no more reliable than the life cycle approach, is no less reliable either. In this regard, perhaps the biggest danger with prototyping, as with any new and highly acclaimed method for doing anything, is that *it may be oversold.* It is not a panacea for all problems of all systems, but it is a viable alternative for development information systems, including library catalog systems, under certain conditions.

In conclusion, the often-quoted adage with regard to democracy may be considered applicable also to prototyping: it may not be perfect, but up to now no one has come up with anything better. It (1) allows freedom to innovate and modify an emerging system to a degree which has never before been possible; (2) involves the user in an intimate relationship with the system from its earliest stages; and (3) permits the construction of a system that meets user requirements, even those initially ill-defined, more closely than any previous method of systems development has done. Maryam Alavi sums up the pluses and minuses of the prototyping approach in one elegant paragraph, which seems an appropriate way to conclude this work as well:

> In summary, the prototyping approach offers an opportunity to achieve favorable user attitudes toward the design process and the information system. Furthermore, it facilitates fast response to user needs, allows clarification of user requirements, and offers an opportunity for experimentation. Although there are pitfalls and shortcomings, none seem troublesome enough to outweigh the benefits.[61]

REFERENCES

1. Wasserman, Anthony I. "Information System Design Methodology." *JASIS* 31(Jan. 1980):5.
2. Ibid., p. 6.
3. Burch, John G., Jr., and Strater, Felix R., Jr. *Information Systems: Theory and Practice.* Santa Barbara, Calif.: Hamilton Publishing Company, 1974, pp. 10-14.

4. Mader, Chris. *Information Systems: Technology, Economics, Applications*. Chicago: Science Research Associates, 1974, p. 327.
5. Wyllys, Ronald E. "System Design—Principles and Techniques." *Annual Review of Information Science and Technology*, vol. 14. White Plains, N.Y.: Knowledge Industry Publications, Inc. for ASIS, 1979, p. 5.
6. Ibid.
7. Heany, Donald F. *Development of Information Systems: What Management Needs to Know*. New York: The Ronald Press, 1968, p. 45.
8. Ibid.
9. Wasserman, "Information System Design Methodology," p. 8.
10. Ibid., p. 9.
11. Gane, Chris, and Sarson, Trish. *Structured Systems Analysis: Tools and Techniques*. Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1979, pp. 4-6.
12. Robinson, C.R., et al., "The Techniques of Systems Analysis." In *Reader in Library Systems Analysis*, edited by John Lubans, Jr. and Edward Chapman, p. 31. Englewood, Calif.: Microcard Editions Books, 1975.
13. Corbin, John. *Developing Computer-Based Library Systems*. Phoenix, Ariz: Oryx Press, 1981, pp. 109-15.
14. Fasana, Paul J. "Systems Analysis." *Library Trends* 21(April 1973):472-75.
15. Ibid., p. 472.
16. Carter, Ruth. "Systems Analysis as a Prelude to Library Automation." *Library Trends* 21(April 1973):505.
17. Fasana, "Systems Analysis," p. 470.
18. Chapman, Edward A. "Planning for Systems Study and Systems Development." *Library Trends* 21(April 1973):483.
19. Carter, "Systems Analysis as a Prelude," p. 513.
20. Griffin, Hillis L. "Implementing the New System: Conversion, Training and Scheduling." *Library Trends* 21(April 1973):573.
21. Wasserman, Anthony I., and Shewmake, David T. "Rapid Prototyping of Interactive Information Systems ACM Sigsoft Software." *Engineering Notes* 7(Dec. 1982):171.
22. Taylor, Tamara, and Standish, Thomas A. "Initial Thoughts on Rapid Prototyping Techniques." *ACM Sigsoft Software Engineering Notes* 7(Dec. 1982):160.
23. Ibid., p. 161.
24. Wasserman, and Shewmake, "Rapid Prototyping," p. 172.
25. Ibid., pp. 171-72.
26. Taylor, and Standish, "Initial Thoughts on Rapid Prototyping Techniques," pp. 160-61.
27. Wasserman, and Shewmake, "Rapid Prototyping," p. 173.
28. Corey, James F., and Bellomy, Fred L. "Determining Requirements for a New System." *Library Trends* 21(April 1973):549-50.
29. Zimmerman, Patricia J. "Principles of Design for Information Systems." *JASIS* 28(July 1977):190.
30. Read, Nigel S., and Harmon, Douglas L. "Assuring MIS Success." *Datamation* 27(Feb. 1981):116.
31. Podolsky, Joseph L. "Horace Builds a Cycle." *Datamation* 23(Nov. 1977):165.
32. Keen, Peter G.W. "Adaptive Design for Decision Support Systems." *ACM Database* 12(Fall 1980):15.
33. MacEwan, Glenn H. "Specification Prototyping." ACM *Sigsoft Software Engineering Notes* 7(Dec. 1982):112.
34. McCoyd, Gerard C., and Mitchell, John R. "System Sketching: The Generation of Rapid Prototypes for Transaction Based Systems." *ACM Sigsoft Software Engineering Notes* 7(Dec. 1982):127.
35. Smoliar, Stephen W. "Approaches to Executable Specifications." *ACM Sigsoft Software Engineering Notes* 7(Dec. 1982):155.
36. Naumann, Justus D., and Jenkins, A. Milton. "Prototyping: The New Paradigm for Systems Development." *MIS Quarterly* 6(Sept. 1982):30.
37. Alavi, Maryam. "The Evolution of Information Systems Development Approach: Some Field Observations." *Database* 15(Spring 1984):21.

38. Keus, Hans E. "Prototyping: A More Reasonable Approach to System Development." ACM *Sigsoft Software Engineering Notes* 7(Dec. 1982):94.

39. Ibid.

40. Earl, Michael J. "Prototype Systems for Accounting, Information and Control." *Accounting Organizations and Society* 3(No. 2, 1978):168.

41. Blum, Bruce I. "Rapid Prototyping of Information Management Systems." *ACM Sigsoft Software Engineering Notes* 7(Dec. 1982):37.

42. Alavi, Maryam. "An Assessment of the Prototyping Approach to Information Systems Development." *Communications of the ACM* 27(June 1984):559.

43. Naumann, and Jenkins, "Prototyping: The New Paradigm for Systems Development," p. 31.

44. Alavi, "The Evolution of Information Systems Development Approach," p. 24.

45. Naumann, and Jenkins, "Prototyping: The New Paradigm for Systems Development," p. 37.

46. Ibid.

47. Date, C.J. *An Introduction to Database Systems,* 3d ed. Reading, Mass.: Addison Wesley, 1981, p. 25.

48. Read, Nigel S., and Harmon, Douglas L. "Assuring MIS Success." *Datamation* 27(Feb. 1981):109.

49. Ibid.

50. Blum, "Rapid Prototyping of Information Management Systems," p. 36.

51. Saffady, William. "Data Management Software for Microcomputers: dBase II." *Library Technology Reports* 19(Sept./Oct. 1983):485-92.

52. Ibid., p. 487.

53. Ibid., p. 489.

54. Wasserman, and Shewmake. "Rapid Prototyping of Interactive Information Systems," Notes 7, p. 173.

55. Alavi, Maryam "An Assessment of the Prototyping Approach," p. 561.

56. Feather, Martin S. "Mapping for Rapid Prototyping." *ACM Sigsoft Software Engineering Notes* 7(Dec. 1982):23.

57. McCoyd, Gerard C., and Mitchell, John M. "System Sketching," p. 130.

58. Mason, R.E.A., et al. "ACT/1: A Tool for Information Systems Prototyping." *ACM Sigsoft Software Engineering Notes* 7(Dec. 1982):120.

59. Blum, "Rapid Prototyping of Information Management Systems," p. 35.

60. Taylor, Tamara, and Standish, Thomas A. "Initial Thoughts on Rapid Prototyping Techniques," p. 166.

61. Alavi, "An Assessment of the Prototyping Approach," p. 563.

ADDITIONAL REFERENCES

Armstrong, C.J. "The Use of Commercial Microcomputer Database Management System as the Basis for Bibliographic Information Retrieval." *Journal of Information Science* 8(June 1984):197-201.

Appleton, Daniel S. "What Data Base Isn't. *Datamation* 23(Jan. 1977):85-92.

Balzer, Robert M., et al. "Operational Specification as the Basis for Rapid Prototyping." *ACM Sigsoft Software Engineering Notes* 7(Dec. 1982):3-16.

Barstow, David. "Rapid Prototyping, Automatic Programming, and Experimental Sciences." *ACM Sigsoft Software Engineering Notes* 7(Dec. 1982):33-34.

Blair, John C., Jr. "Creating Your Own Database." *Database* 5(Aug. 1982):11-17.

Bonet, Rafael, and Kung, Antonio. "Structuring into Subsystems: the Experience of a Prototyping Approach." *ACM Sigsoft Software Engineering Notes* 9(Oct. 1984):23-27.

Bordwell, Stephen. "dBase II—Library Use of a Microcomputer Database Management System." *Program* 18(April 1984):157-65.

Brill, Evan L. "ACTS—An Application-Generator for Building Online Transaction Systems." In *Proceedings of the ASIS Annual Meeting,* vol. 19, edited by Anthony E. Petrarca, Celianna I. Taylor, Robert S. Kohn, pp. 43-46. White Plains, N.Y.: Knowledge Industry for ASIS, 1982.

Burns, R.W., Jr. "A Generalized Methodology for Library Systems Analysis." *College & Research Libraries* 32(July 1971):295-305.

Carr, Marilyn. "Build Your Own Database: Using Database Management Systems for Custom Applications." In *Proceedings of the Fourth National Online Meeting* (New York, 12-14 April 1983), compiled by Martha E. Williams and Thomas H. Hogan, pp. 87-92. Medford, N.J.: Learned Information, 1983.

Chandor, Anthony, et al. *Practical Systems Analysis.* London: Rupert Hart-Davis Educational Publications, 1969.

Chapman, Edward A., et al. *Library Systems Analysis Guidelines.* New York: Wiley-Interscience, 1970.

Clinton, Marshall. "Phoenix: An Online System for a Library Catalogue." *Database* 5(Feb. 1982):52-65.

Cohen, Donald, et al. "Using Symbol Execution to Characterize Behavior." *ACM Sigsoft Software Engineering Notes* 7(Dec. 1982):25-32.

Corbin, John. *Developing Computer-Based Library Systems.* Phoenix, Ariz.: Oryx Press, 1981.

Costa, Betty. *A Micro Handbook for Small Libraries and Media Centers.* Littleton, Colo.: Libraries Unlimited, 1983.

Crawford, R.G., and Macleod, I.A. "A Relational Approach to Modular Information Retrieval Systems Design." In *The Information Age in Perspective* (Proceedings of the ASIS Annual Meeting, 1978), vol. 15, compiled by E.H. Brenner, pp. 83-85. White Plains, N.Y.: Knowledge Industry for ASIS, 1978.

Davis, Alan M. "Rapid Prototyping Using Executable Requirements Specification." *ACM Sigsoft Software Engineering Notes* 7(Dec. 1982):39-44.

Debons, Anthony, and Montgomery, K. Leon. "Design and Evaluation of Information Systems." *Annual Review of Information Science and Technology.* Washington, D.C.: ASIS, 1974, vol. 9, pp. 25-55.

Dixon, John K., et al. "Rapid Prototyping by Means of Abstract Module Specifications Written as Trace Axioms." *ACM Sigsoft Software Engineering Notes* 7(Dec. 1982):45-49.

Dodd, W.P. "Prototype Programs." *Computer* 13(Feb. 1980):81.

Dodd, W.P., et al. "Prototyping Language for Test-Processing Applications." *ACM Sigsoft Software Engineering Notes* 7(Dec. 1982):50-53.

Fitzgerald, John M., and Fitzgerald, Ardra F. *Fundamentals of Systems Analysis.* New York: John Wiley & Sons, 1973.

Ford, Ray, and Marlin, Chris. "Implementation Prototypes in the Development of Programming Language Features." *ACM Sigsoft Software Engineering Notes* 7(Dec. 1982):61-66.

Gehani, N.H. "A Study in Prototyping." *ACM Sigsoft Software Engineering Notes* 7(Dec. 1982):71-74.

Gibson, F., and Nolan, R.S. "Managing the Four Stages of EDP Growth." *Harvard Business Review* 52(Jan./Feb. 1974):76-88.

Gill, Hans, et al. "Experience from Computer Supported Prototyping for Information Flow in Hospitals." *ACM Sigsoft Software Engineering Notes* 7(Dec. 1982):67-70.

Gillespie, Jim. "dBase II at Nepean, Ont., Public Library." *Canadian Library Journal* 41(Dec. 1984):339-43.

Goguen, Joseph, and Meseguer, Jose. "Rapid Prototyping in the OBJ Executable Specification Language." *ACM Sigsoft Software Engineering Notes* 7(Dec. 1982):75-84.

Gorry, G. Anthony, and Scott Morton, Michael S. "A Framework for Management Information Systems." *Sloan Management Review* 12(Fall 1971):55-70.

Gregory, S.T. "On Prototypes vs. Mockups." *ACM Sigsoft Software Engineering Notes* 9(Oct. 1984):13.

Groner, Gabriel F., et al. "Requirements Analysis in Clinical Research Information Processing—A Case Study." *Computer* 12(Sept. 1979):100-08.

Hayes, Robert M., and Becker, Joseph. *Handbook of Data Processing for Libraries*, 2d ed. Los Angeles, Calif.: Melville Publishing Co., 1974.

Heitmeyer, C., et al. "The Use of Quick Prototypes in the Secure Military Message Systems Project." *ACM Sigsoft Software Engineering Notes* 7(Dec. 1982):85-87.

Hice, Gerald F., et al. *System Development Methodology*, rev. ed. Amsterdam: North-Holland Publishing Co., 1978.

Hooper, James W., and Hsia, Pei. "Scenario-Based Prototyping for Requirements Identification." *ACM Sigsoft Software Engineering Notes* 7(Dec. 1982):88-93.

Huffenberger, Michael A., and Wigington, Ronald L. "Database Management Systems." *Annual Review of Information Science and Technology*. White Plains, N.Y.: Knowledge Industry Publications, Inc. for ASIS, 1979, vol. 14, pp. 153-90.

Klausner, A., and Konchan, T.E. "Rapid Prototyping and Requirements Specifications Using PDS." *ACM Sigsoft Software Engineering Notes* 7(Dec. 1982):96-105.

Lehman, M.M. "Research Proposal to Study the Role of Executable Metric Models in the Programming Process." *ACM Sigsoft Software Engineering Notes* 7(Dec. 1982):106-11.

Liston, David M., Jr., and Schoene, Mary L. "A Systems Approach to the Design of Information Systems." In *Key Papers in the Design and Evaluation of Information Systems*, edited by Donald W. King, pp. 327-34. White Plains, N.Y.: Knowledge Industry for ASIS, 1978.

MacKenzie, A. Graham. "Systems Analysis as a Decision-Making Tool for the Library Manager." *Library Trends* 21(April 1973):493-504.

Markuson, Barbara Evans, et al. *Guidelines for Library Automation: A Handbook for Federal and Other Libraries*. Santa Monica, Calif.: Systems Development Corp., 1977.

Minder, Thomas. "Application of Systems Analysis in Designing of a New System." *Library Trends* 21(April 1973):553-64.

Mittermeir, Roland T. "HIBOL, A Language for Fast Prototyping in Data Processing Environments." *ACM Sigsoft Software Engineering Notes* 7(Dec. 1982):133-40.

Montgomery, David B., and Urban, Glen L. "Marketing Decision-Information Systems: An Emerging View." *Journal of Marketing Research* 7(May 1970):226-34.

Norman, Margaret. *Considerations in Planning an In House Database Retrieval System* (Proceedings of the 7th International Online Information Meeting, London, 6-8 Dec. 1983). Oxford: Learned Information, 1983, pp. 213-17.

Podolsky, Joseph L. "Horace Builds a Cycle." *Datamation* 23(Nov. 1977):162-68.

Ramanthan, J. "Use of Annotated Schemes for Developing Prototyping Programs." *ACM Sigsoft Software Engineering Notes* 7(Dec. 1982):141-49.

Rao, V. Venkata. "Some Approaches to Modeling Complex Information Systems." *Information Processing & Management* 18(No. 3, 1982):151-60.

Read, Nigel S., and Harmon, Douglas L. "Assuring MIS Success." *Datamation* 27(Feb. 1981):109-20.

Rich, Charles, and Waters, Richard C. "The Disciplined Use of Simplifying Assumptions." *ACM Sigsoft Software Engineering Notes* 7(Dec. 1982):150-54.

Robinson, C.R., et al. "The Techniques of Systems Analysis." In *Reader in Library Systems Analysis*, edited by John Lubans, Jr. and Edward A. Chapman, pp. 29-52. Englewood, Calif.: Microcard Editions Books, 1975.

Rosove, Perry E. "Introduction." In *Developing Computer-Based Information Systems*, edited by Perry E. Rosove, pp. 67-93. New York: John Wiley & Sons, Inc., 1967.

Rowley, J.E. *Mechanised In-House Information Systems*. London: Clive Bingley, 1979.

Saffady, William. "Data Management Software for Microcomputers: dBASE II." *Library Technology Reports* 19(Sept./Oct. 1983):485-95.

Scheffler, Frederic L. "Novel Philosophy for the Design of Information Storage and Retrieval Systems Appropriate for the 70's." *JASIS* 24(May/June 1973):205-09.

Smith, David. *Systems Thinking in Library and Information Management*. New York: K.G. Saur, 1980.

Smith, David Andrew. "Rapid Software Prototyping." Ph.D. diss., University of California, Irvine, 1982.

Smoliar, Stephen W. "Approaches to Executable Specifications." *ACM Sigsoft Software Engineering Notes* 7(Dec. 1982):155-59.

Squires, Stephen L., et al. "Rapid Prototyping Workshop: Overview." *ACM Sigsoft Software Engineering Notes* 7(Dec. 1982):2.

Stavely, Allan M. "Models as Executable Designs." *ACM Sigsoft Software Engineering Notes* 7(Dec. 1982):167-68.

Stow, William R., and Vogt, Earl C. "Prototype Development of Legislative Management Information System." In *The Information Age in Perspective* (Proceedings of the ASIS Annual Meeting, 1978), vol. 15, pp. 330-32. White Plains, N.Y.: Knowledge Industry for ASIS, 1978.

Strand, Eugene M., and Jones, Warren T. "Prototyping and Small Scale Software Projects." *ACM Sigsoft Software Engineering Notes* 7(Dec. 1982):169-70.

Sullivan, Jeanette. "Using dBASE II for Bibliographic Files." *Online* 9(Jan. 1985).

Taylor, Tamara, and Standish, Thomas A. "Initial Thoughts on Rapid Prototyping Techniques." *ACM Sigsoft Software Engineering Notes* 7(Dec. 1982):160-66.

Vickery, B.C. *Information Systems.* London: Butterworths, 1973.

Wasserman, Anthony I. "Information System Design Methodology." *JASIS* 31(Jan. 1980):5-24.

Weber, Herbert. "The Distributed System—A Monolithic Software Development Environment." *ACM Sigsoft Software Engineering Notes* 9(Oct. 1984):43-73.

Weiser, Mark. "Scale Models and Rapid Prototyping." *ACM Sigsoft Software Engineering Notes* 7(Dec. 1982):181-85.

Werner, David J., et al. "Designing Rational Systems." In *Key Papers in the Design and Evaluation of Information Systems*, pp. 345-56.

Wyllys, Ronald E. "System Design—Principles and Techniques." *Annual Review of Information Science and Technology*, vol. 14. White Plains, N.Y.: Knowledge Industry Publications, Inc. for ASIS, 1979, pp. 3-35.

Zelkowitz, Marvin V. "A Taxonomy of Prototype Designs." *ACM Sigsoft Software Engineering Notes* 9(Oct. 1984):11-12.

VITAE

SUSAN SMERNOFF LAZINGER

Susan Smernoff Lazinger is Head of the library of the Graduate School of Library and Archive Studies, Hebrew University, Jerusalem, Israel. She is currently on leave as special project cataloger of the Taube-Baron Collection of Jewish History and Culture at Stanford University

Dr. Lazinger has a Ph.D. in Slavic Languages and Literatures from Syracuse University and an M.L.S. with a specialization in information science from Hebrew University. She worked in the History of Science Collections at the Jewish National and University Library, Jerusalem, for several years and has also worked in the Syracuse University Library.

In addition to her publications in the field of Russian literature, Dr. Lazinger's previous publications in the field of library and information science include "The Pitt Study and Its Critics: A Survey of the Literature" (in Hebrew with English summary), Yad-Lakore *The Reader's Aid: Israel Journal for Libraries and Archives* 22(No. 2, 1985):41-47; "LC Classification of a Library and Information Science Library for Maximum Shelf Retrieval." *Cataloging and Classifying Quarterly* 5(Winter 1984):45-50; "Tobin's 'A Study of Library Use Studies': An ERIC Update of Neglected Areas of Research." *Government Publications Review* 11(1984):165-71; and "The Alexandrian Library and the Beginnings of Chemistry." *Library History Review* 2(Sept. 1975):36-47 [actually published in 1984]. She is on the editorial boards of ISLIC *Bulletin* (a publication of The Israel Society of Special Libraries and Information Centres) and *Judaica Librarian.*

PERETZ SHOVAL

Peretz Shoval is on the faculty of the Department of Industrial Engineering & Management and the Department of Computer Science at Ben-Gurion University of the Negev. He is also affiliated with the Graduate School of Business Administration at Tel-Aviv University and with the Hebrew-University of Jerusalem. He received a B.A. in Economics; an M.S. in Information-Systems from Tel-Aviv University, and a Ph.D. in Management-Information-Systems from the University of Pittsburgh. Dr. Shoval did his thesis on expert systems for information retrieval.

Dr. Shoval's research and teaching interests include systems analysis & design methodologies, database design, expert systems, and information retrieval. He has published in journals such as *Information Systems, Information Processing & Management, International Journal of Man-Machine Studies, Data and Knowledge Engineering, Information & Management,* and *Data Base.*