The impact of implementing a rapid prototype on system maintenance

by JOHN L. CONNELL
Martin Marietta Denver Aerospace
Denver, Colorado

and

LINDA BRICE
Los Alamos National Laboratory
Los Alamos, New Mexico

ABSTRACT

Rapid prototyping is a popular new way of developing software applications in both business and scientific environments. The technique is invaluable in providing communication between customers and developers; but like any other methodology, it can fail if not used properly. This paper compares and contrasts two case studies of prototyping endeavors. One appears to be a success, and the other had major problems and may result in total failure. The reasons for the phenomena are probed, concluding with an observed successful prototyping life cycle, which, if abandoned, may negate the positive aspects of the entire prototyping process. The necessary phases of rapid analysis, database development, menu development, function development, prototype iteration, detailed design, and implementation have been well documented. It is strongly suggested that three new phases—planning, modeling, and training—be added to the cycle for a better chance of successful rapid prototyping.
IMPLEMENTING THE PROTOTYPE: TWO CASE STUDIES

Much attention has been devoted to the techniques of rapid prototyping in recent literature. Some organizations have been convinced that the idea is viable and have begun to use it as a development tool. Often, when rapid prototyping is promoted, the promotion is accompanied by some caveats, such as “be prepared to throw the prototype away” and “do not attempt to implement a prototype; detailed design will still need to be carefully prepared.” This paper describes two development projects in which rapid prototyping was employed. One succeeded and one failed. In both cases the standard caveats were specified, and during both projects some surprises occurred. The psychology of some of the situations was that the cautions were intellectually considered, but not internalized or fully accepted. From the experience gained during the development of these two prototypes, some of the caveats have become recognized as simply too important to be ignored under any circumstances. Ignoring them has clearly become a sure way to make the implementation of a prototype fail.

The following material illustrates the critical differences in the techniques used to develop and implement each of the systems. The lessons learned from this comparison are then generalized into guidelines for success and absolutes for avoiding failure in rapid prototyping. One conclusion that must be emphasized at the beginning: It may often be the case that the rapid prototype itself is done correctly, but the surrounding life cycle phases are not, and that this is the primary cause of failure, not the techniques of rapid prototyping. There is a danger that too many failures in the application of a new technique will cause critics to rise up in force and insist on its abolition.

For both systems INGRES, a relational database management system, was used to develop a rapid prototype according to methods described in the literature. The authors observed prototype development in separate organizations. In both cases the systems were implemented by persons who were not closely involved in prototype development. Both prototypes were good ones in that they were working models that were developed rapidly and met with user approval. One became a usable system, but the other may be abandoned, for reasons that will be described.

The Resource Control System Prototype: a Failure

This prototype was developed at Company A with the intention of quickly replacing an existing system, which, it had been determined, did not adequately meet user requirements and was difficult to modify. The purpose of the application was to capture data on the cost of development, modification, and operation for all software application systems. This information would then be reported to system developers, project managers, organization management, and end users. The system was to be interactive and allow for accurate forecasting of effort and comparisons of forecasts with actual costs. The primary users for this system were the software developers themselves.

Software developers at Company A are expected to fill out timesheets weekly, showing how many hours were worked on various system phases. These data, along with hardware operation costs, are the primary input of cost information into the system. Monthly forecasts of these costs are also required and input into the system. The system must then produce a large variety of reports: forecast effort, weekly timesheets, costs by system, costs by user, and many more. These reports have multiple uses, such as forecasting tools, customer billing reports, and hiring justifications. Figure 1, a dataflow diagram, gives a high-level view of the inputs and outputs for the resource control system.

The useful purposes for information from this system are almost unlimited. Unfortunately, the old system did not provide this information in an acceptably accurate, timely, or
user-friendly fashion. Therefore it was decided to use rapid prototyping techniques to model possible resource-efficient solutions. A new system was to be brought up quickly on a low budget. This project could be considered a failure, since the old system may have to be continued and the prototype scrapped.

The Comprehensive System Development Environment Prototype: a Success

This prototype was developed at Company B with the intention of helping to define user requirements for a set of new automated software development tools. It is a brand new application in the sense that an automated system performing the same functions did not exist previously. Its purpose was to provide, through the use of a common project database, a set of tools to help analysts, designers, programmers, project managers, and other project support personnel with the preparation of software system specifications and documentation. Figure 2 is a visual representation of the comprehensive system development environment. Creation of the total system will be a very elaborate project, originally estimated to require about 50 person-years of development effort.

Management decided to use rapid prototyping to develop an interim product: a requirements document generator. In this way it was felt that, through demonstrations of the system to potential users, design details of the common project database and the user interface could be worked out most efficiently. The requirements document generator is based on the assumption that a requirements analyst will be using structured analysis incorporating dataflow diagrams, process descriptions, and a data dictionary. Further, it is assumed that the printed requirements document will be in military standard format. The prototype was developed to allow information generated during structured analysis to be stored in the common project database in a way that would allow these data to be combined with standard boilerplate text and formatted correctly into a requirements document that would meet with customer approval. Figure 3 is a high-level dataflow diagram showing how the requirements generator interfaces with other parts of the comprehensive system development environment.

One of the problems frequently encountered in using structured analysis correctly is the amount of manual effort involved in checking for errors or violations of structured-analysis guidelines. Therefore, consistency-checking procedures were built into this prototype. These procedures use the features of a relational database to compare the analysis information entered against a set of guidelines and output a set of reports listing the violations. The prototype then became a tool to assist analysts with structured analysis, rather than simply a document formatter and generator. As a test of the workability of the system, the requirements for the comprehensive system development environment itself were entered into the prototype, and a requirements document was generated in military standard format.

It was found that the resulting requirements document was prepared in an unusually short period of time, with much higher than usual accuracy, because of the consistency-checking features of the prototype. Problems encountered during preparation of the document caused refinements to be made to the prototype, resulting in a more robust system. Finally it was discovered that some of the functional modules developed for the requirements document generator could be used with slight modifications in the development of other system functions, such as the design document generator. This project succeeded, yet prototyping was carried out in much the same fashion as for the failure cited above. There were some setbacks, but they were not due to prototyping. They probably would have happened anyway, and because the project was developed as a prototype, they were more easily
corrected. The differences in project management between the two case studies will be discussed to show the reason for the difference in outcome.

THE (REVISED) LIFE CYCLE OF A RAPID PROTOTYPE

The authors have published prototyping life cycle phases in previous works. On the basis of recent experience gained from the projects just described, new phasing elements are believed to be necessary in the prototyping life cycle. The previous version of the life cycle called for a project to begin with a phase called rapid analysis, in which preliminary user interviews were to be used to develop an intentionally incomplete, high-level, paper model of the system to be prototyped. A first-cut prototype would then be created with a relational database structure, developing menus and functions using fourth-generation techniques to produce a working model. Next a phase called prototype iteration is entered, in which the user is shown the prototype using familiar data. User suggestions for improvements are incorporated into successive iterations until the working model provides complete satisfaction. It is then suggested that detailed design be derived from the user-approved prototype to adequately document the system for maintenance purposes. Finally, it is recommended that an implementation checklist be used to ascertain that the prototype has been officially approved and that all documentation is complete and accurate.

These are good procedures to follow in rapid prototyping. However, on the basis of the case studies, there are three phases that should be added: planning, as the first phase; modeling, or benchmarking, as a phase to follow prototype iteration; and training of system maintenance staff. Figure 4 shows all phases of the new, revised life cycle model, with appropriate feedback and feed-forward of information indicated by arrows. The reasons for the addition of the three new phases should be apparent from the following material.

A COMPARISON OF LIFE CYCLE ACTIVITIES

Planning the Failure

A project plan, partially shown in Figure 5, was produced. It was correct in format and approved by a walkthrough team consisting of peers. In reality, the project plan was not followed. The budget was cut drastically, both in staff and in schedule. For example, whereas the project plan forecast a demonstrable prototype in October and historical data loaded by experienced staff members, in actuality the prototype was demonstrated in January and implemented in August, complete with historical data, and only inexperienced student labor was used. Why was the budget so severely modified? There were a number of reasons, and not one of them was due to improper motivation. There was an optimistic belief that just maybe this one time the odds could be beaten because so much was at stake. Management was in an untenable situation: an internal system was needed, but proper staffing was difficult to justify, since the new system would have an unclear payoff, at least in the early stages. Student employment was available only for the summer, combined with only one staff member who was terminating at the same time. The belief, the real hope, was that some of the system could be implemented before the staff melted away. In that way, corporate memory could be preserved, the major functions could be used, and some time could be bought to staff the system properly for completion of implementation and for maintenance.

As has often been documented, cutting budgets is almost always unwise and was probably the single most obvious reason in this case for system failure. How does budget cutting differ in a prototyping environment from the traditional development environment? In several ways. One is that, because visible outputs are quickly available, users and managers are easily seduced into believing that the design/code/test phase as well as the modeling phase can be skimped on. Even mature, experienced management and analysts must be especially careful in realizing that menus and screens are not backed up with detailed design and that reports may be perfectly formatted, using data elements carefully defined, yet still suffer tremendously in the performance area. The point is that, as in the case of the failure, even a correctly prepared project plan that is not adhered to will accomplish nothing.
Planning the Success

A project plan was completed. It was actually improved upon, because the schedule was originally based on traditional development methods calling for a much longer development cycle than was needed for rapid prototyping. The original estimate was 50 person-years; the projected estimate now that one particular phase has been successfully prototyped and implemented is 10 person-years. Since the original plan called for detailed design to be derived from the approved prototype, the design was completed despite arguments that it was now unnecessary because the prototype worked so well. The prototype was not implemented without carrying out all the steps called for in the project plan. It is obviously important not just to prepare a project plan, but to follow the plan during development.

Rapid Analysis: the Success

The analysis phase of the successful prototype was quite problematic. First, the system concept was so new that neither users nor developers could offer a clear definition of system requirements. There were some vague notions of what an ideal system would do, but serious misgivings about how many of these functions it would be practical to implement. Second, none of the persons who were performing the analysis had any formal training in structured analysis. Finally, the walkthroughs of the analysis documents were very poorly structured and frequently degenerated into all-day oral defenses of the document by its author.

The result was that when users finally had a chance to review the analysis document, they became confused and angered. They were confused because the paper model did not clearly represent the proposed system requirements. They were angered because what they could understand implied that the system to be developed would be of little use to them. Fortunately, prototype development was under way by the time the ensuing argument between users and analysts reached its peak. Just as the user group was beginning to clamor for development funds to be cut, a demonstrable prototype was ready. When the users saw the demonstration of the prototype, they were pleasantly surprised to discover that it contained all the functions it would be practical to implement. The strengths and weaknesses of the old system were clearly represented in the paper model, and users found that it was easy to understand. Thus, the user group was pleased, and they made a second cut in development funds because they were so pleased with the prototype.

Rapid Analysis: the Failure

The analysis of the system was not terribly difficult, because the system under development was based on an existing system. The existing system was clumsy to use and difficult to maintain because it was old, little analysis had been performed on it when it was first developed, it was COBOL/sequential file/ISAM file/batch based, and it suffered from hard-coded data. It produced only fragmented information for a small segment of the user community. Over the years, the users had developed definite opinions about what their needs really were, where the functions of the system should be expanded, and how the input/output cycles should work within the framework of corporate policy. The rapid-analysis phase proceeded with ease; and although no product can be considered perfect, it probably would look very much the same if it were redone with different analysis staff, just because the strengths and weaknesses of the old system were already known. A paper model was produced, using dataflow diagrams for the system context and essential functions models. The paper model included an entity relationship diagram showing relationships among aggregations of data elements or entities.

This high-level design was probably adequate. The fact that the system would become a failure was not apparent during the Rapid Analysis phase. The paper models were useful during prototyping and the database structure, and functional development followed the design of the models.

Figure 5—Project Plan

Project Plan

<table>
<thead>
<tr>
<th>PROJECT JUSTIFICATION</th>
<th>The system will provide actual employee effort, reporting, forecasts, computer costs and other project management information to management and customers...</th>
</tr>
</thead>
<tbody>
<tr>
<td>GOAL</td>
<td>The creation of an interactive system which can be used to house and report actual person effort expended on each project undertaken by the division...</td>
</tr>
<tr>
<td>SCOPE OF WORK</td>
<td>1. High-level functional analysis... 2. Developed prototype... 3. Design of physical model... 4. Refined prototype... 5. Final system documentation... 6. Estimated effort...</td>
</tr>
<tr>
<td>USER RESPONSIBILITIES</td>
<td>1. The walkthrough team will be responsible for approving the prototype... 2. Division management will be responsible for staffing the data entry duties...</td>
</tr>
<tr>
<td>DELIVERABLE PRODUCT</td>
<td>1. The prototype will a) be a database system... 2. The implemented system will a) include historical data... b) include enhancements for equipment...</td>
</tr>
<tr>
<td>SCHEDULE</td>
<td>1. Project plan approved. 01/23/84 2. Functional analysis. 02/01/84 3. Prototype operational. 05/01/84 4. User Quick Guide. 06/01/84 5. System Design. 08/01/84 6. Historical data loaded. 02/15/85</td>
</tr>
</tbody>
</table>
Prototype Development: Both Cases

Prototype development consists of three steps: database development, menu development, and function development. These steps were carried out in similar fashion for both systems. First a basic database structure was developed on the basis of the paper models. Next a menu structure was created as a user interface, using a screen-based forms tool provided by the relational database management system. Finally, functions driven by the database system were developed. In both cases the writing of third-generation programs was avoided, and a working model was available very quickly.

It is difficult to make unrecoverable mistakes during these phases. The prototype is based on a relational database and has no lines of code, and it is therefore very easy to modify. In fact, since the main purpose is to find out whether the working model pleases the user, many changes are to be expected. That being the case, it is not possible to prove that one prototype is better than another. They are all simply first cuts—intended to be refined, enhanced, tuned, tested, and possibly even thrown away if the original concept proves unfeasible. It is true that in the case of the failure, an inexperienced staff was busy developing a system that would not function very smoothly in a production mode. But then their charter was to produce a rapid prototype, not a production system.

Certainly elements of the failure can be detected in this phase. The quickening pace of the schedule prevented thorough consideration of the volume of data and how those data would need to be linked for outputs. For example, even though data tables are nicely normalized, too many joins may make the table structure viable. Elements of the success can also be detected in the prototype development. The developers of the successful prototype realized early in the game that as soon as users saw a useful working model, there would arise a great clamor for its implementation as a production system. Therefore these developers purposely built nonworking functional modules into the system. These modules looked nice during demonstrations but prevented the system from being used in production.

Iterating the Success

The successful prototype was never released to users in a hands-on mode. Demonstrations were provided where developers operated the prototype and users were simply the audience. The only people who were allowed to use the prototype were the developers themselves. Large groups of users reviewed the prototype in several of these demonstrations, and their comments were used to refine the system. Thus, by giving users the feel of how useful the implemented prototype would be, the developers obtained user feedback without raising expectations too high. When the occasional user demanded immediate implementation, the nonworking modules mentioned earlier effectively prevented it from happening.

The result of this rigid stance on the part of the developers was that implementation was delayed until a smoothly operating, bug-free system could be delivered. Since this was accomplished in much less time than was previously thought possible, rapid prototyping acquired a reputation for being a better way to develop application systems. High-level managers within the company are now proud to claim that Company B has mastered this state-of-the-art technique.

Modeling or Benchmarking

This is one of the newly recommended phases. It is possible that prototype iteration has provided functionality with which users are perfectly satisfied, but the system still won’t work. The modeling phase consists of performance testing to determine response time characteristics of the system in a production environment. It does not involve user interaction, but it can be critical for many different types of applications. The system must be exercised with expected maximum usage and data loads before it can be determined that the current incarnation is implementable. This is particularly true of relational database systems, where it is possible to create database structures and functional procedures that will work adequately under small loads but slow down unacceptably under large loads.

There is not much to say about this phase in the case of the failure; it simply didn’t happen. The prototype was implemented, and its undesirable performance characteristics were
discovered during production operations. On the other hand, the successful prototype was used by developers in a realistic production emulation mode to actually perform all its functions on large amounts of real data. As a result, the successful prototype went into a rather lengthy phase of elaborate tuning and refinement. Tables were radically restructured, and some procedures that had been developed rapidly with fourth-generation techniques were replaced with modules developed using lower-level, but more efficient, procedures. In this way a pleasing demonstration-only prototype was converted into a production-quality system.

**Designing the Failure**

The failure did not go undocumented. A very professional, accurate, and understandable user guide was produced. This was accomplished with relative ease by printing input and output screens and menus and adding explanatory dialogue. Missing from the final documentation was a detailed design of system procedures and control flows. High-level analysis is fine for prototyping, but next to useless for maintenance work or system tuning.

For the system to be easy for anyone to maintain, detailed descriptions of its architecture and control sequences need to be published. Unfortunately, this was one of the steps that was skipped in the rush to implement. Later, when it was discovered that there were some problems in running the prototype in a production mode, these problems were not easy to remedy without the proper design documentation. If the project plan had been adhered to, it would have been noted that an extensive detailed design phase was called for. Remember that the project plan was agreed upon by users, managers, developers, and their peers. The organizational wisdom at the time had determined that the design effort called for was absolutely critical to project success.

**Designing the Success**

The data that were entered into the successful prototype and used to produce output consisted of detailed specification of the prototype itself. All functional modules were completely specified to the lowest levels, complete with accurate data definitions. This detailed design was derived from the latest user-approved version of the prototype. During the last part of the benchmarking phase and into implementation, a change in development staff personnel made this documentation extremely valuable. Since the specification now accurately describes the implemented system in complete detail, a new maintenance person can learn the system well enough to make needed changes in less than a week.

**Implementing the Failure**

It is surprising that the failure was allowed to be put into production without the developers' first completing an implementation checklist, considering that the organization where the failure occurred usually adheres strictly to rigid implementation standards that explicitly call for such a checklist. Perhaps, since this system was developed at the request of internal users rather than external customers, it was thought that this was a step that could be eliminated in the interests of desired budget cutting. Implementation checklists are simply a way of determining whether the project completed all activities and delivered all products called for by the project plan and internal standards. Since the failure clearly did not, publishing an implementation checklist would have advertised this embarrassment.

Implementing the failure consisted of an abrupt switch from use of the old system to use of the prototype. This is the point at which those responsible for maintaining the prototype began to complain loudly and bitterly. They said that the table structures were all wrong, that coding standards had been violated, and that the lack of adequate design made modifications next to impossible. Poor performance characteristics of the prototype gave them all the ammunition they needed to get the prototype scrapped and return to using the old system.

**Implementing the Success**

The successful prototype was also implemented without benefit of a formal checklist. The organization simply does not use such a device, relying instead on elaborate review procedures during earlier development phases. At the point where the system test plan has been reviewed and approved and the proposed system subjected to it successfully, the system is delivered to the customer. In this case, however, the prototype developers kept their own checklist and refused to let the system be implemented until it had satisfied all the criteria defined in the project plan for successful implementation.

**Maintaining the Failure**

The maintenance phase was the most difficult in the case of the failure. It will be one of the causes in the case of the prototype's final demise. The maintenance staff personnel were not familiar with rapid prototyping techniques and had been loudly skeptical, all during development, of the advisability of using these techniques. Now they had an opportunity to say "I told you so." Courses offering instruction in rapid prototyping techniques had been offered within the organization, but later discontinued. Previous graduates of these courses were available, but none of them was assigned to maintain this prototype. Management's thinking may have been that, although training may be helpful for developing rapid prototypes, it is not needed for maintaining them.

A rapid prototype involves the application of fairly radical techniques incorporating fourth-generation language and relational database tools. Maintenance personnel with a third-generation coding background will be hard pressed to understand a system developed with these techniques unless they are given adequate training. In a highly technical field, lack of understanding is threatening to one's professional stature; there is a tendency to belittle techniques with which one is not familiar.
Maintaining the Success

Even though the staff responsible for implementing and maintaining the system consisted of personnel completely different from the development staff, they had been thoroughly trained and believed in the techniques of rapid prototyping. Remember that only one interim product from an extensive proposed tool suite was implemented. The remainder of the tools will be developed in rapid prototype fashion by the current staff. They are enthusiastic because they have discovered that the modular style of rapid prototype development leads to reusable and replaceable functional modules. This will make their future development work much easier than the development of the original prototype.

Examining the Critical Differences

Figure 6 lists the differences between the life cycle activities of the two projects. For each area where a difference occurred, there is a brief descriptive statement. It can be observed that there are no significant differences applicable to the development of the prototypes themselves. The conclusion that can be drawn is that one project was managed well and the other was mismanaged. In addition, there are also some obvious techniques for making a prototype successful, rather than for making a good prototype.

CONCLUSIONS DRAWN

One might ask, in the case of the failure, why the mature project team members deviated so far from years of experience-based knowledge. The answer is an important one; prototyping is still new as an industry tool, and we are prone to misuse it. It is the most valuable of all the tools of communication between customer and developer. It solves the problem of delivering the wrong system better than any other thing else we know of at this time. It is, however, potentially hazardous because it can persuade all of us, both customer and developer, that now that we have successfully defined what is desired and necessary in a system, implementation will be trivial. While identifying the problems is a large part of the battle in any problem-solving situation, it is by no means the entire activity.

Defining Rapid Prototyping

During the prototype iteration phase, functions may be redefined; menus may be added, deleted or modified; and, more likely than not, the database structure may be altered (although this activity will probably peak during modeling when restructuring is done for performance reasons). Once again, the point must be made that the system is not fully fleshed out at this point and should not be implemented. In the case of the failure, a great deal was lost by putting the system on the floor for everyone to use and test. The skeptics criticized every minor problem as though it were a fully tested system instead of a refined prototype, and the proponents liked it so much they were incredulous that the system might be scrapped. Much goodwill was sacrificed in both camps because prototype was improperly defined for the entire customer base and because there was no “friendly user phase.”

Following a Sensible Plan

A sensible project plan is one that includes all the activities listed in Figure 4 with realistic estimates of the time required to complete them. A peer review of the plan will help to assure reasonableness. If users or management want to revise the plan or stray from its directives, a new plan with new deliverables should be published. If the project plan was deemed reasonable at the start of a project, budget cuts during development must result in reduced deliverables if implementation is to be successful. If the prototype development team is unsure that the current project plan is being adhered to, they should insist on the publication of a checklist prior to implementation.

Training for Rapid Prototyping

In the case of the failure there was a broad-based misconception on the part of many users, managers, and proposed maintenance staff about the definition and meaning of rapid prototyping. To reiterate the common confusion, rapid prototyping is a requirements analysis technique, not a system development technique. These misconceptions also exist in the organization that implemented the successful system; but they did not become critical because they were not held by key managers, developers, or maintenance staff. The best way to clear up misconceptions about rapid prototyping is with an in-house training program. Unfortunately, such training is not readily available from commercial sources. A possible method of implementing such a program within an
organization is to select a successful rapid prototyper and place him or her in charge of designing the training program. If no such person is available, then it might be wise to reconsider the advisability of using rapid prototyping techniques. The misuse of rapid prototyping, as illustrated by the failure, can yield more disadvantages than the benefits to be gained from its proper application.

REFERENCES