Prototyping: alternative systems development methodology

J M Carey

Prototyping has become a popular alternative to traditional systems development methodologies. The paper explores the various definitions of prototyping to determine its advantages and disadvantages and to present a systematic methodology for incorporating the prototyping process into the existing system development process within an organization. In addition, one negative and one positive case study of prototyping within industrial settings is included.

system development methodologies, prototyping, software life-cycle

In recent years, use of prototyping has increased dramatically for both the requirements definition phase of the systems development life-cycle and rapid building of end-user systems[1]. The increase has been primarily due to the advent of fourth-generation language (4GL) application generators.

A study of Texas-based computer facilities showed that prototyping was more widely used than almost any offline, structured, software-development tools, such as dataflow diagrams and decision tables[1].

This paper explores the definition of prototyping, the advantages and disadvantages of using this technique, and how to determine when a prototyping approach is appropriate.

CONSENSUS DEFINITION

If various analysts and programmers were asked to define prototyping, the responses would vary considerably, depending on experience and training. Prototyping has taken on a variety of meanings and uses and has been variously defined as follows:

> 'a strategy for determining requirements wherein user needs are extracted, presented, and defined by building a working model of the ultimate system – quickly and in context' (p 25)[2]

> 'Prototyping is based on building a model of the system to be developed. The initial model should include the major program modules, the data base, screens, reports and inputs and outputs that the system will use for communicating with other, interface systems' (p 69)[3]

> 'working models used to check accuracy of designs before committing to full-scale production' (p 79)[4]

> 'The idea behind prototyping is to include users in the development cycle' (p 93)[5]

What do these definitions have in common? First, prototyping is seen as a model of the final system, much like in the automobile industry where prototype or model cars are built and tested before full-scale production is attempted. In prototyping a software system, only parts of the system are developed, with a key emphasis on the user interfaces, such as menus, screens, reports, and source documents. The prototype is then a shell of the final system with no calculations and data behind the interfaces. The final system is either built from scratch using the prototype as a model or evolved from the prototype.

Second, the emphasis is on user involvement in the software development process. In the traditional software development life-cycle, communication between analysts and users occurs early in the cycle to determine information needs, then the analysts work, in isolation, to develop the system and seldom interact with the users until system delivery and production. As users have little input into the development process, the resultant system is often dissatisfactory and difficult to learn and use. Prototyping provides a 'hands-on' communication tool to allow the analyst to determine user needs and ensure ongoing communication throughout the development process, thus ensuring that the system is the 'right' one for the user.

Third, prototyping produces an information system faster than using the traditional life-cycle approach. When users are frustrated by the development backlog that exists in most organizations, speed of delivery can be a great selling point. This is often called 'rapid prototyping' by proponents and 'quick and dirty' by opponents.

Taking these three underlying ideas and incorporating them into one gives the following consensus definition:

> 'Prototyping' is the process of quickly building a model of the final software system, which is used

Arizona State University – West Campus, P.O. Box 37100, Phoenix, AZ 85069–7100, USA.

Paper submitted: 19 April 1989.
Revised version received: 12 September 1989.

Reprinted from *Information and Software Technology*, Vol. 32, No. 2, March 1990, pp. 119–126.

primarily as a communication tool to assess and meet the information needs of the user.

RATIONALE FOR PROTOTYPING

The traditional software development approach has several inherent problems, which prototyping attempts to address. These problems include the following[2,6]:

- Users seldom have clear, concise understanding of their informational needs. Therefore, they cannot prespecify the requirements. Once they begin to use a system, however, it is clear to them where the problems lie.
- The traditional function specification is a narrative description of an information system that is technical and time consuming to read. Static graphic techniques (such as dataflow diagrams, and data dictionary entries found in the structured approach) once thought to be the solution to communication cannot demonstrate the workings of a live dynamic system[7].
- The larger the development team, including user representatives, the more difficult communication becomes[8]. Semantic barriers and lack of physical proximity and time inhibit the ability of all members of the team to have a common understanding of the system being developed.
- Even if systems developed in the traditional manner function correctly, they may be difficult to learn and use.
- Both traditional and structured approaches emphasize documentation, which is time consuming and as the system changes may not be accurate[9].
- Systems being developed today are more complex, have a larger mission, and require many months to complete. The traditional approach has not served to shorten delivery time, in fact it may unduly lengthen the time required due to the emphasis on documentation[9].
- Because of the large number of people/months involved and time-consuming methods, traditional approaches not only seem to deliver late systems that do not please the user, they are also costly.
- Most large companies have a long backlog of projects awaiting initiation, while the users who requested them are frustrated, disillusioned, and ready to revolt.

All of these problems suggest that some revolutionary technique is needed. Prototyping is one technique that attempts to address these problems and provide possible solutions.

PROTOTYPING ENVIRONMENTS

There are two major types of prototyping environments[5,10]. One is a complete and integrated application-generator environment or automated development environment (ADE), which can produce quick, integrated menus, reports, and screens and is tied to a database. Examples are R:base 5000 or System V for the microcomputer and NOMAD2 for the mainframe.

A prototyping toolkit comprises the other environment. The toolkit is a collection of unintegrated tools that aid the rapid building of the separate pieces of a system, such as screen painters, data dictionaries, and report generators. Together, these tools are often referred to as analysts' or programmers' 'workbench'.

The following 'workbench' tools can aid the prototyping process:

- text editors
- screen generators
- report generators
- relational databases
- fourth-generation languages (4GLs)
- spreadsheets
- data dictionaries coupled to database management systems
- *ad hoc* query languages
- security
- statistical packages
- back-up routines
- documentation generators
- online help
- interactive testing system

If purchased separately, these tools are initially expensive when compared with the traditional method of coding in a third-generation language (3GL) such as COBOL. Also, before jumping into prototyping, a training period for both development team and users is required.

Acquiring the tools or environment is just the first step. Once the environment for building a prototype has been created and staff and users thoroughly trained in the use of prototyping tools, a systematic methodology should be adopted that is tailored to the specific organization and then followed to ensure that the system that results from the prototyping technique is both usable and correct. All too often, companies purchase prototyping packages and jump into prototyping without trying to determine when and how to use the technique.

The following five steps are suggested by Klinger[6], manager of laboratory systems and programming at Ortho Pharmaceutical Corporation, as a successful approach to the use of prototyping:

- Assess each application individually. Would prototyping provide gains?
- Look at the environment and then develop and document a formal prototyping life-cycle that fits it.
- Acquire appropriate software tools and train the staff.
- Decide how the software development process will be managed and controlled.
- Train end-users in the procedures that will be followed during the prototyping life-cycle.

ITERATIVE (TYPE I) VERSUS THROWAWAY (TYPE II) PROTOTYPING

One confusion in defining prototyping arises from the existence of two distinct types of prototyping that are used by various companies. These two basic approaches to prototyping are iterative and throwaway. The iterative approach (Type I) uses the prototype as the final system after a series of evolutionary changes based on user feedback. The throwaway approach (Type II) uses the prototype built in a 4GL as a model for the final system, with the final system coded in a 3GL.

In the Type I (iterative) approach, the life-cycle consists of the following stages[6]:

- training
- project planning
- rapid analysis
- database development
- prototype iteration
- modelling
- detailed design
- implementation
- maintenance

The inclusion of training and project planning is unique. These stages are seldom mentioned in the traditional life-cycle. The modelling stage is also unique and important. It is at this stage that the prototype system is tested through benchmarking to make sure it performs within acceptable standards. Possible replacement code may be needed at bottlenecks in the prototype. Sometimes 3GL code may be substituted for any original 4GL that has been determined as inefficient. Figure 1 shows the system development life-cycle incorporating Type I prototyping.

In the Type II (throwaway) approach, some iteration occurs and the steps of analysis, design, coding, testing, and modification may be repeated many times until all of the users' requirements are identified and met. Once the prototyping phase is complete, then the prototype serves as a model for final production system, but is discarded at the project delivery[6]. The throwaway prototyping approach generally adheres to the traditional life-cycle once the prototype has been developed. Figure 2 illustrates the system development life-cycle incorporating the Type II prototyping technique.

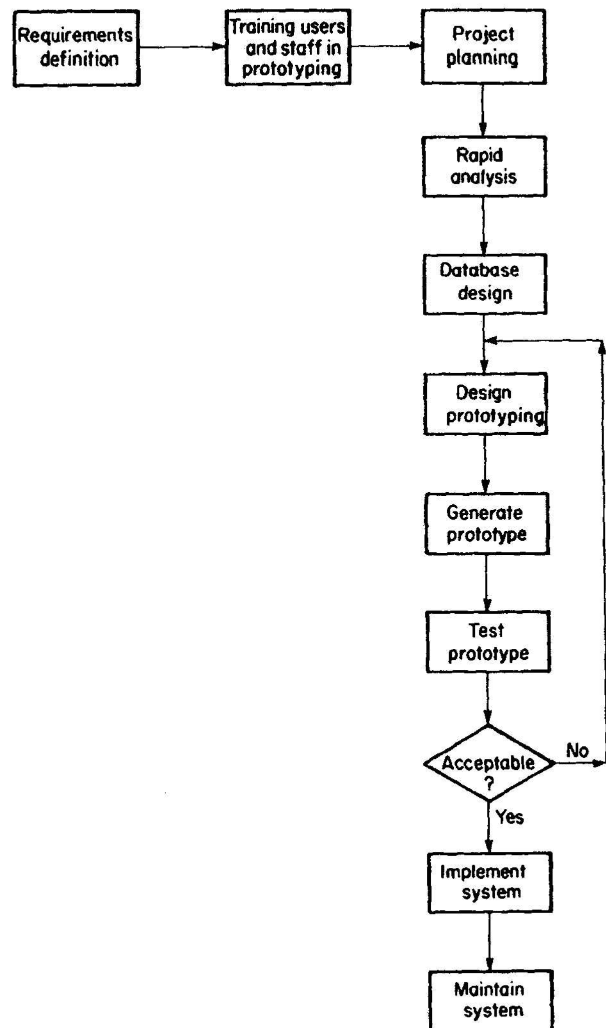

Figure 1. System development life-cycle using Type I (iterative) prototyping

ADVANTAGES OF PROTOTYPING

Prototyping is being used in industry with varying degrees of success. Proponents of prototyping cite the following positive attributes:

- Systems can be developed much faster[11].
- Systems are easier for end-users to learn and use.
- Programming and analysis effort is much less (less humanpower needed).
- Development backlogs can be decreased[12].
- Prototyping facilitates end-user involvement.
- System implementation is easier because users know what to expect.
- Prototyping enhances user/analyst communication.
- User requirements are easier to determine.
- Development costs are reduced.
- The resultant system is the 'right' system and needs little changing.

All of these positive attributes make prototyping sound like the system development dream, like the answer to all analyst's and user's problems. Indeed, many organizations have adapted some use of prototyping within their development life-cycle. However, there is a downside to prototyping.

DISADVANTAGES OF PROTOTYPING

Undue user expectations[13] The ability of the systems group to develop a prototype so quickly may raise undue expectations on the part of the user. They see the shell and may not understand that it is not the finished system. They may have been waiting for this system for months or even years and are so anxious to get something in

place that being so close and yet so far may frustrate them even more.

Inconsistencies between prototype and final system If the prototype is a throwaway type, the end system may not be exactly like the prototype. In other words, what the user sees may not be what the user gets. It is up to the analyst to communicate any differences between the prototype and the end system; if the user is forewarned, the negative reaction may be ameliorated. It is advisable to ensure that the resultant system be as close to the prototype as possible to avoid this potential problem.

Encouragement of end-user computing The availability of prototyping software both in the organization and on the general market may encourage end-users to begin to develop their own systems when their needs are not being met by data-processing staff. While end-user involvement in system development is positive, end-user computing (development of systems by end-users) may have some negative ramifications for system integration and database integrity.

Final system inefficiencies[14] Large, complex systems that require voluminous numbers of transactions may not be good candidates for the iterative prototyping technique. 4GLs have a reputation for generating less than optimal code in terms of efficiency and throughput. Care must be taken to predetermine whether the new system should be written with an application generator/ prototyping tool or prototyped in a 4GL and then coded in a 3GL for maximum efficiency. A discussion of how to make these determinations is included in the next section.

Lack of attention to good human factors[5] The use of application generators as prototyping tools does not ensure that the resultant systems will adhere to human-factors guidelines. In fact, many application generators have rather inflexible screen and menu formats, which often inhibit the use of good human-factors techniques unless additional background code is written (defeating the purpose of the application generator).

Inattention to proper analysis Because prototyping application generators are relatively easy to use and produce quick results, analysts are tempted to plunge into prototyping before sufficient analysis has taken place. This may result in a system that looks good, with adequate user interfaces, but that is not truly functional. This is how the reputation of 'quick and dirty' prototypes came about. To avoid this pitfall, a well defined methodology that stipulates the stages of prototyping is necessary.

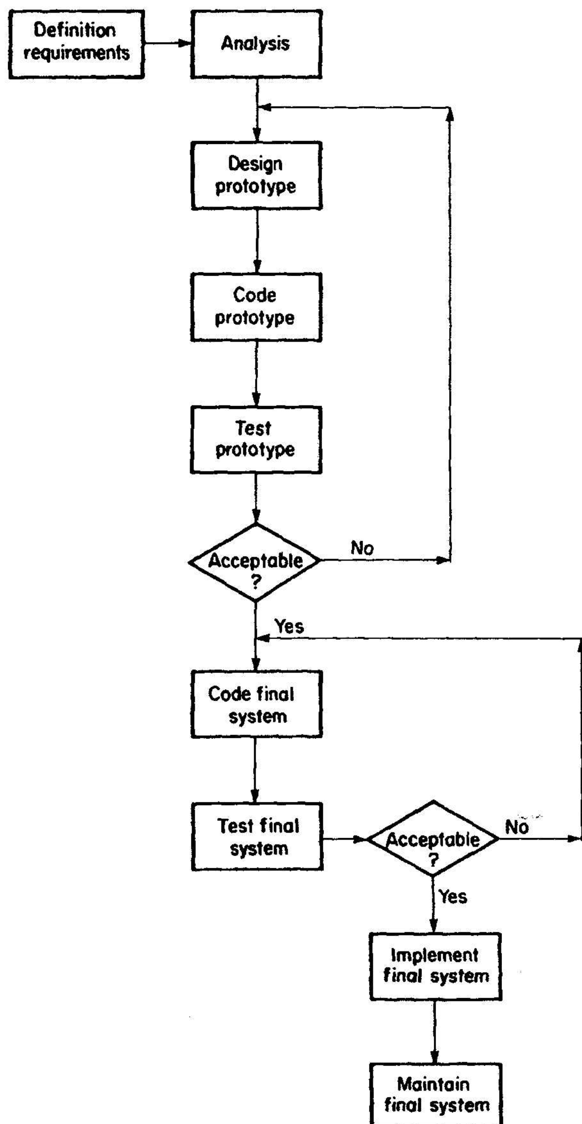

Figure 2. Traditional system development life-cycle with Type II (throwaway) prototyping

DETERMINATION OF WHEN TO PROTOTYPE

Some form of prototyping may be used in the development of all systems from large and complex to small and simple. Determination of whether to use the iterative prototyping technique, which will evolve into the final system, or the throwaway type, which may be used primarily to model the user interfaces, however, is dependent on several variables.

If the system in question has the following characteristics, it may be a prime candidate for iterative prototyping[3,6]:

- is dynamic (always changing)
- is transaction-processing based
- contains extensive user dialogues
- is small versus large
- is well defined
- is online
- 'is' the business (i.e., billing, record management, transaction-driven, predetermined structure)

On the other hand, if the system exhibits the following characteristics, iterative prototyping is unlikely to enhance the final system[3,6]:

- is stable
- is decision-support based

- contains much *ad hoc* retrieval and reporting
- is of no predictable form
- is ill defined
- is batch
- makes little use of user dialogues
- is large and complex
- is real-time
- does extensive number crunching
- is 'about' the business rather than directly involved in transaction processing (i.e., decision support and expert systems)

METHODOLOGY

Type I or Type II prototyping can be effectively used when developing information systems; the key to success is carefully determining which prototype type to use and then following a well defined methodology.

The methodology should include thorough requirements definition and design stages before any prototyping is attempted. The prototype should then be defined, coded, tested, and used to refine the requirements and design and put to use as a Type I or Type II prototype. During the refinement process, user comments and responses can be solicited and used to alter any unsatisfactory portions of the prototype. Once the user(s) and analyst are satisfied with the prototype, then the prototype can either be retained and expanded to become the final system or used as a model for the final system that is developed in a 3GL.

There are four phases that are inherent in the development and completion of a prototype[15].

Determination of key aspects of system to be prototyped
The three main areas that are often prototyped include the user interface, uncertain or vague system functions, and time and memory requirements. Any or all three of these aspects can be prototyped.

User interface The most common area to be prototyped. Many prototyping tools are specifically aimed at rapid development of menus, screens, and reports. This is the aspect that the user must understand and accept for the system to be successful.

Uncertain system functions Often, the development of a new system includes some functional processing that may not be well understood by any team members. This uncertain area is a probable candidate for prototyping. The development of a working model allows the team to make sure that the solution they are proposing will indeed satisfy the requirements and perform effectively. The involvement of the user will not be as heavy for this type of prototype as for the user interface. The user may not fully understand the calculations and output. The user may be able to provide both test input and output data, however, to verify the model.

Time and memory requirements The exercise of these aspects may be more appropriately termed a simulation instead of a prototype. Many systems may be characterized by huge volumes of transactions and data manipulations. Standards for interactive response times and memory use can be established, and the prototype/simulation is exercised to ensure that the system can accomplish the functional tasks within the standards range.

Building the prototype
Many tools are available for building prototypes, as already mentioned. The prototype is initially built quite rapidly using one or more of the prototyping tools.

Testing the prototype
The prototype is tested and debugged based on user and performance feedback.

Using the prototype as a model
The prototype is used as a model for the final system (Type II) or as the base for the final system (Type I).

Adherence to a strict methodology will help to ensure the success of the prototyping approach and will combat the 'quick and dirty' system development that sometimes results from prototyping in a haphazard manner.

INCORPORATING HUMAN-FACTORS GUIDELINES INTO PROTOTYPING

Even though prototyping provides an excellent method of analyst/user communication, there is nothing inherent in the prototyping tools to ensure adherence to good human-factors guidelines. Therefore, analyst/programmers should have additional training in this critical area. Human factors in systems and the issue of 'user friendliness' or 'usability' has been recognised recently as a determinant of system success. Just because a system is technically sound does not mean that it will be easy to learn and use. The following human-factors guidelines[16] should be adhered to as part of the system design phase.

Know your users Users today range from novice to expert. There are many variables that can help profile users, including previous exposure to computers, the nature of the task they are attempting to perform on the system, level of training, how often they use the system in question, level in the organization, amount of dependency on the computer, etc. One of the first tasks of the analyst should be to profile the user population.

Use selection not entry Whenever possible, allow the user to select information from possible options on the screen rather than require the user to remember what to do next. Humans forget and have enough task variables in their short-term memory to worry about without having to memorize how to get the system to function. The only problem with selection not entry is that it may slow up the experienced user. In order not to frustrate this type of user, program selections to accept multiple keystrokes that will allow the experienced user to sidestep the selection process.

Make the system behave predictably Consistent design of function keys and options will lead to ease of learning and use. Switching and interchanging will lead to frustration and abandonment of the system.

Make the system as unobtrusive as possible The focus of any computer session should be on the work task rather than on the system itself. Some aspects of the interface, such as blinking, reverse video, colour use, and audibles can be distracting rather than meaningful, especially when the user is doing routine data entry and has to be involved with the system on an extended daily basis. These attention-getting devices may be helpful as 'training wheels' during the learning process, but probably should be removed once the user is 'up' on the system.
Use display inertia when carrying out user requests The display should change as little as possible. This helps to prevent user distraction.
Conserve muscle power A single keystroke or depression of a function key is usually faster and less cumbersome than multiple keystrokes, particularly for the intermittent user who is not a proficient typist.
Use meaningful error messages If the user makes a mistake, advise on what the mistake was and how to correct it. Avoid negative, patronizing messages. Simply state the problem and how to correct it.
Allow for reversing of actions Protect the users from the system and the system from the users. Create a suspense file that can be altered and verified before the database is altered; this will help to ensure database integrity. Allow failsafe exits from the system at any time.

This list is not all inclusive, of course. There are many other sources for user-interface design guidelines[17]. Incorporating these guidelines into interface designs and using prototypes to communicate user requirements will help to ensure system success.

TWO CASE STUDIES WITH PROTOTYPING

Case 1: New Jersey Division of Motor Vehicles[14]

From 1983 to 1985, the State of New Jersey Division of Motor Vehicles contracted Price Waterhouse and Company to build its primary information system. A new 4GL named Ideal from Applied Data Research (ADR) Inc. was used to develop the system. When the system was delivered, the response times were so slow that the backlogs generated from using the system resulted in thousands of motorists driving with invalid registrations or licences. Overtime pay to employees amounted to hundreds of thousands of dollars. In short, at delivery time, the system was declared a total disaster.

Why was Ideal chosen as the language for this development project? First, time pressures dictated speedy completion of the project and, second, the Systems and Communications (SAC) Division of the State of New Jersey had already acquired ADR's Datacom/DB, which supported Ideal as a 4GL. The decision was made by Price Waterhouse to use Ideal, against the recommendations of several members of the SAC.

Robert Moybohm, the SAC's deputy director, had earlier evaluated Ideal for possible use in other, smaller projects and determined:

- Ideal would not be able to handle the development of large, online systems. He ran some benchmark tests against COBOL programs and Ideal ran three times slower on simple processing.
- Ideal did not offer index processing, a performance-related feature that had been the initial reason that SAC purchased the Datacom/DB system in the first place.
- Ideal did not allow computer-to-computer interfacing. The large system would need to interface with 59 other computers. This fact alone should have precluded the selection of Ideal.

Why did Price Waterhouse choose Ideal? What went wrong? From the beginning, poor decisions were made about the system development process. Ideal was a brand-new product and was not well tested. The development staff had no experience using any 4GL and considerable time was spent learning and making mistakes. All along the development cycle, it became apparent that the system was not going to meet performance requirements, yet no one was able to stop the process and change to a 3GL or determine how to combat the performance problems. It seems that one of the driving forces was the fact that the development team was locked into a fixed-cost contract and delivery date and that every month after deadline would incur a stiff financial penalty. So a decision was made to deliver a nonperforming system within the deadline rather than a late, but functional, one.

After failed implementation of the new system and the resultant flurry of irate users died down, an attempt was made to rectify the problem. It was determined that only about 58 of the 800 program modules needed to be converted to COBOL to meet acceptable, response-time criteria. Eight modules were responsible for the nightly batch updates. The other 50 modules were online programs that were handling 85% of the system's transaction volume. It was not merely a simple line-by-line conversion; many modules had to be redesigned to achieve performance requirements.

The impact of a failed system on the motorists of New Jersey could have been avoided by running the old system in parallel with the new system until the problems were rectified. Instead, due primarily to costs and inadequate hardware resources, direct cutover implementation was used as a strategy. Consequently, the failure of the new system was evident to everyone in the state of New Jersey, not just to internal staff.

Was Type I (iterative) prototyping with a 4GL the wrong choice for the New Jersey Division of Motor Vehicles? Given the volume of transactions, and the development team's inexperience, the answer must be yes. A more effective approach would have been to use the Type II (throwaway) approach, using the 4GL to model the system, rather than use the Type I (iterative) approach to develop the end system.

Case 2: Town and Country Credit Line (TCCL)

In early 1988, Town and Country Credit Line (TCCL) decided to develop a system to enhance their competitive advantage over other banking cards. TCCL has long seen itself as the leader in banking card technology. (The actual nature of the system is proprietary at this time and the name of the company has been changed.) TCCL decided to explore the costs and benefits of using CASE technology to enhance delivery time for new systems. They chose a service request system as an eight-week pilot project to accomplish this purpose. They hired outside programmer/consultants who had experience in the use of CASE technology and purchased IEF (Information Engineering Facility) from Texas Instruments.

The decision to develop the service request system as a pilot was based on the following:

- the estimated short time required to deliver this product to the user community (it was perceived to be a system with a fairly narrow scope)
- the time the user community had been promised the system with no delivery
- it was felt that this system would give the development team the 'biggest bang for the dollar' (quote from project manager)
- it was felt that this system would provide the user community with a system that would dramatically enhance productivity while simplifying complex choices

Why did they not just use traditional methods to develop this system? The system had features that they felt would be very difficult to design and produce using traditional methods. These features were interprocedure communication and linking of procedures.

Because CASE technology was new to the organization, the pilot project would additionally serve to provide a knowledge base within the team to make accurate estimates for projects that use the CASE tool, and give each team member an opportunity to gain 'hands-on' experience with all phases of the CASE tool, and in doing so provide understanding of the limitations and capabilities of the CASE tool.

Two consultants were hired to provide support during the pilot project as the team had no experience with IEF. One consultant provided guidance on the methodology and project management, the other on IEF itself.

Training of the resident staff members was limited. At the beginning of the pilot project, only two team members out of nine had any training beyond Business Area Analysis and Business System Design. No team members had any training or experience with IEF technical design and construction. One team member had no training in CASE and IEF at all.

The system was developed by breaking it into its logical business components and then distributing one task to each group. The system was developed within the eight-week deadline and performs the required tasks efficiently and effectively with user acceptance. Two problems were encountered during the development process. One was related to the CASE technology and the other to the nature of the system. As the CASE technology was new to the organization, a learning curve was encountered. The competence of the team and a willingness to work additional hours helped to overcome this problem. The other problem was a lack of communication between groups. The groups sometimes went off on inconsistent tangents and some work had to be redone. Once the product is familiar to the team, and less time is spent on learning IEF, scheduled full-team meetings could alleviate this problem.

IEF divides the development process into seven steps:

- ISP (Information Strategy Planning). Allows identification of areas of concern and establishment of direction.
- BAA (Business Area Analysis). Areas of concern are analysed for entity relationship and process dependency.
- BSD (Business System Design). The processes are packaged into procedures that are user interactive.
- TD (Technical Design). The conversion of BAA/BSD designs into specific database tables (such as DB2), CICS transactions, and COBOL II code.
- Construction. The generation of source and executable code and database definition and access statements.
- Transition. Loading data into the databases, determination of conversion strategies.
- Production. Actual implementation and ongoing use of the system.

Throughout these phases, testing also occurs. Unit or program testing is performed by individual team members. System testing occurs when the entire system is operational. User acceptance testing occurs at various points in the development process.

Some problems occurred with the interfaces between systems. Once these problems were solved, the end system performed adequately in terms of efficiency and effectiveness measures. The users were pleased with the system and it is currently functional.

Why was this prototyping effort successful, whereas the effort made by the New Jersey Division of Motor Vehicles unsuccessful? One of the main advantages is five years of advancement in prototyping tools. Ideal is less integrated and much less sophisticated than IEF. Also TCCL has had a chance to learn from other companies' mistakes. As prototyping software tools become more and more sophisticated, the inefficiencies will be reduced dramatically.

SUMMARY

Prototyping is the process of quickly building a model of the final software system, which is used primarily as a communication tool to assess and meet the information needs of the user.

Prototyping came about with the advent of 4GLs, which enabled application or code generation. The rea-

sons for the success of prototyping arise from the problems encountered in the use of the traditional development of software systems using 3GLs.

Prototyping environments are divided into two major types: complete application generator environments and toolkit or 'workbench' environments.

There are two major types of prototyping approaches: iterative (Type I) and throwaway (Type II). In the iterative approach, the prototype is changed and modified according to user requirements until the prototype evolves into the final system. In the throwaway approach, the prototype serves as a model for the final system, which is eventually coded in a 3GL or procedural language.

Some advantages of prototyping include: faster development time, easier end use and learning, less humanpower to develop systems, decreased backlogs, and enhanced user/analyst communication. Some disadvantages of prototyping include: the fostering of undue expectations on the part of the user, what the user sees may not be what the user gets, and availability of application-generator software may encourage end-user computing.

Not all systems are good candidates for the prototyping approach. Care should be taken to determine whether the system in question exhibits characteristics that make prototyping a viable option.

No current prototyping tools ensure that good human-factors guidelines will be exhibited in the final system. Analysts should be aware of these guidelines and build systems that adhere to them, regardless of the use of prototyping tools.

Prototyping is a powerful and widely used approach to system development. Systems built with the use of prototyping can be highly successful if a strict methodology is adhered to and thorough analysis and requirements definition takes place before prototyping is attempted.

REFERENCES

1 **Carey, J M and McLeod, Jr, R** 'Use of system develoment methodology and tools' *J. Syst. Manage.* Vol 39 No 3 (1987) pp 30–35

2 **Boar, B** 'Application prototyping: a life cycle perspective' *J. Syst. Manage.* Vol 37 (1986) pp 25–31

3 **Lantz, K** 'The prototyping methodology: designing right the first time' *Computerworld* Vol 20 (1986) pp 69–74

4 **Staff** 'The next generation' *Banker* Vol 136 (1986) pp 79–81

5 **Stahl, B** 'The trouble with application generators' *Datamation* Vol 32 (1986) pp 93–94

6 **Klinger, D E** 'Rapid prototyping revisited' *Datamation* Vol 32 (1986) pp 131–132

7 **Yourdon, E** *Managing the structured techniques* Yourdon Press, New York, NY, USA (1976)

8 **Brooks, F P** 'The mythical man-month' (chapter 2) in **Brooks, F P** (ed) *The mythical man-month essays on software engineering* Addison-Wesley, Reading, MA, USA (1979) pp 11–26

9 **Boehm, B W** 'Structured programming: problems, pitfalls, and payoffs' *TRW Software Series TRW-SS-76-06* TRW Defence Systems, Redondo Beach, CA, USA (1976)

10 **Sprague, R H and McNurlin, B C** *Information systems management in practice* Prentice Hall, Englewood Cliffs, NJ, USA (1986)

11 **Boehm, B W** *IEEE Trans. Soft. Eng.* (1984)

12 **Goyette, R** 'Fourth generation systems soothe end user unrest' *Data Manage.* Vol 24 (1986) pp 30–32

13 **Kull, D** 'Designs on development' *Computer Decisions* Vol 17 (1985) pp 86–88

14 **Kull, D** 'Anatomy of a 4GL disaster' *Computer Decisions* Vol 18 (1986) pp 58–65

15 **Harrison, T S** 'Techniques and issues in rapid prototyping' *J. Syst. Manage.* Vol 36 (1985) pp 8–13

16 **Sena, J A and Smith, M L** 'Applying software engineering principles to the user application interface' (chapter 6) in **Carey, J M (ed)** *Human factors in management information systems* Ablex, Norwood, NJ, USA (1988) pp 103–116

17 **Shneiderman, B** *Designing the user interface* Addison-Wesley, Reading, MA, USA (1986)

BIBLIOGRAPHY

Doke, E R and Myers, L A 'The 4GL: on its way to becoming an industry standard?' *Data Manage.* Vol 25 (1987) pp 10–12

Duncan, M 'But what about quality?' *Datamation* Vol 32 (1986) pp 135–6

Staff 'Why software prototyping works' *Datamation* Vol 33 (1987) pp 97–103