SOFTWARE PROTOTYPING: TODAY'S APPROACH TO INFORMATION SYSTEMS DESIGN AND DEVELOPMENT

ORLAND LARSON
HEWLETT-PACKARD

Among the challenges facing the data processing community are the increasing costs and time associated with developing applications, the increasing backlog of applications, the excessive time spent maintaining applications, and the shortage of EDP professionals. In addition, systems implementation and functionality are impaired due to the lack of tools which involve end-users in the system development process.

Meeting these challenges requires a more progressive approach to applications development - one that is significantly different from traditional system development cycles. This approach is called SOFTWARE PROTOTYPING.

This paper defines software prototyping, identifies its major uses, reviews the step-by-step prototype development process, and discusses the resources and skills required to effectively prototype applications. It also addresses the problems and costs associated with software prototyping.

INTRODUCTION

The Changing Role of Data Processing

The data processing department has changed dramatically since the 1960's, when application development as well as production jobs were usually run in a batch environment with long turnaround times and out-of-date results.

The 1970's were a period of tremendous improvement for the data processing environment. One of the key developments of that period was the development and use of Data Base Management Systems (DBMS). This provided the basis for on line interactive applications. In addition, computers and operating systems provided programmers the capability of developing application pro- grams on line, sitting at a terminal and interactively developing, compiling, and testing these applications. The end user was also provided with easy to use on-line inquiry facilities to allow them to access and report on data residing in their data bases. This took some of the load off the programmers and allowed them to concentrate on more complex problems.

During the 1980's, for the Data Base Administrator and MIS manager, we see increased importance and use of centralized data dictionaries or "centralized repositories of information about the corporate data resources." We also see simpler and more powerful report writers for the end user and business professional. For the programmer, we see the use of very high level transaction processing languages to reduce the amount of code required to develop applications. Finally, the tools have been developed to effectively do software prototyping which will provide benefits to the end user as well as the application programmer and analyst.

Throughout the Seventies and Eighties, information has become more accurate, reliable, and available, and the end user or business professional is becoming more involved in the application development process.

Challenges Facing MIS

The MIS manager's number one problem is the shortage of EDP specialists. A recent Computerworld article predicted that by 1990 there will be 1/3 of a programmer available for each computer delivered in this country. Software costs are also increasing because people costs are going up and because of the shortage of skilled EDP specialists. The typical MIS manager is experiencing an average of two to five years of application backlog. This doesn't include the "invisible backlog", the needed applications which aren't even

requested because of the current known backlog. In addition, another problem facing MIS management is the limited centralized control of information resources.

The programmer/analyst is frustrated by the changeability of users' application requirements (the only thing constant in a user environment is change). A significant amount of programmers' time is spent changing and maintaining users' applications (as much as 60% of their time). Much of the code the programmer generates is the same type of routines such as error checking, formatting reports, reading files, checking error conditions, data validation, etc. This can become very monotonous or counter- productive for the programmer.

The end user or business professional is frustrated by the limited access to information needed to effectively do his/her day-to-day job. This is especially true for those users who know their company has spent a great deal of money on computer resources and haven't experienced the benefits. The user's business environment is changing dynamically and they feel MIS should keep up with these changes. MIS, on the other hand, is having a difficult time keeping up with these requests for application maintenance because of the backlog of applications and the shortage of EDP specialists. Once the user has "signed off" on an application, he is expected to live with it for awhile. He is frustrated when he requests what he thinks is a "simple change" and MIS takes weeks or months to make that change.

Traditional Approach to Application Development

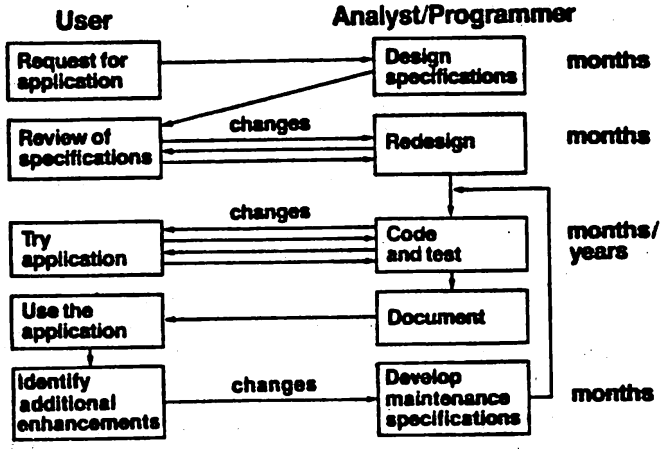
There are some myths concerning application development:

- Users know what they want
- Users can communicate their needs to MIS
- Users needs are static

The traditional approach to application development has serious limitations when applied to on-line, interactive information systems that are in a state of constant change and growth. Communications among the user, analyst, programmer, and manager tend to be imprecise, a detailed analysis prolongs the process to the annoyance of the user, and specifications are either ambiguous or too voluminous to read. To compound this problem, the user is often requested to "freeze" his requirements and subsequent attempts at change are resisted.

Let's review the traditional approach to application development.

TRADITIONAL APPROACH TO APPLICATION DEVELOPMENT



- The user first requests an application and then an analyst or programmer is assigned to the application.
- The analyst or programmer takes the oftentimes sketchy user specifications and designs more complete specifications.

- The user then reviews the analyst's interpretations of his specifications and probably makes additional changes.
- The analyst redesigns his specifications to adapt to these changes. (By this time, several days, weeks or months have gone by.)
- The user approves the specifications and a team of analysts and programmers are assigned to develop, test and document the application. (This may take months or years.)
- The user finally tries the application. Months or years may have gone by before the user gets his first look at the actual working application.

The question is: "Can MIS afford to continue using this traditional approach to application development?"

Prototyping Defined

According to Webster's Dictionary, the term prototype has three possible meanings:

1) It is an original or model on which something is patterned: an archetype.

2) A thing that exhibits the essential features of a later type.

3) A standard or typical example.

J. David Naumann and A. Milton Jenkins in a paper on software prototyping (see reference 3) believe that all three descriptions apply to systems development. Systems are developed as patterns or archetypes and are modified or enhanced for later distribution to multiple users. "A thing that exhibits the essential features of a later type" is the most appropriate definition because such prototypes are a first attempt at a design which generally is then extended and enhanced.

- The user, of course, will want additional changes or enhancements made to the application, to adjust the application to the "real world".
- Depending on the extent of these changes, additional maintenance specifications may have to be written and then coding, testing and documentation.
- The total application development process may take months or years and the maintenance of these applications may go on forever.

Software Prototypes

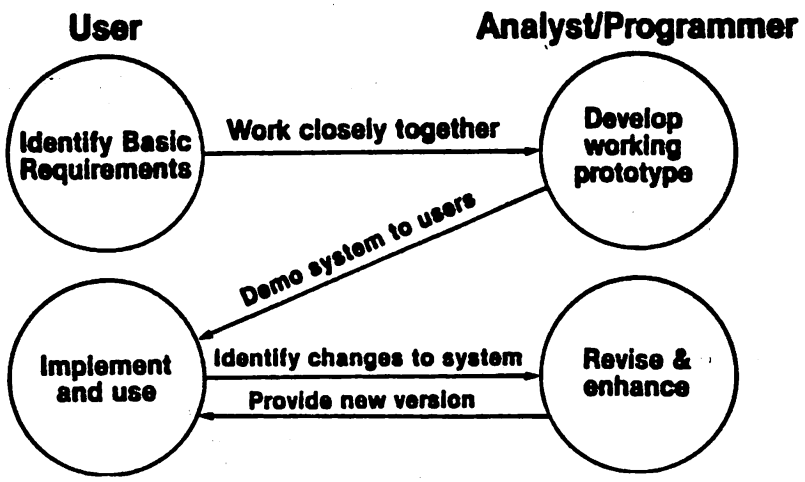
The process of software prototyping is a quick and relatively inexpensive process of developing and testing an application system. It involves the end user and programmer/analyst working closely to develop the application. It is a live, working system; it is not just an idea on paper. It performs actual work; it does not just simulate that work. It can be used to test out assumptions about users' requirements, system design, or perhaps even the logic of a program.

Prototyping is an iterative process. It begins with a simple prototype that performs only a few of the basic functions of a system. It is a trial and error process - build a version of the prototype, use it, evaluate it, then revise it or start over on a new version, and so on. Each version performs more of the desired functions and in an increasingly efficient manner. It may, in fact, become the actual production system. It is a technique that minimizes the dangers of a long formal analysis and increases the likelihood of a successful implementation.

The Prototype Model

Prototyping an information system can be viewed as a four step procedure.

PROTOTYPING APPROACH TO APPLICATION DEVELOPMENT



Step 1. Identify users' basic requirements:

- End user and programmer/analyst work closely together.
- Concentrate on users' most basic and essential requirements.
- Define data requirements, report formats, screens, and menus.
- Need not involve written specifications.
- For larger systems, a design team may need to spend a few weeks preparing a first-effort requirements document.

Step 2. Develop a working prototype:

- Programmer analyst takes the notes developed in the user discussions and quickly creates a working system.
- Designs and/or defines data base and loads subset of data.
- Makes use of defaults and standard report formats.
- Performs only the most important, identified functions.

Step 3. Implement and use the prototype:

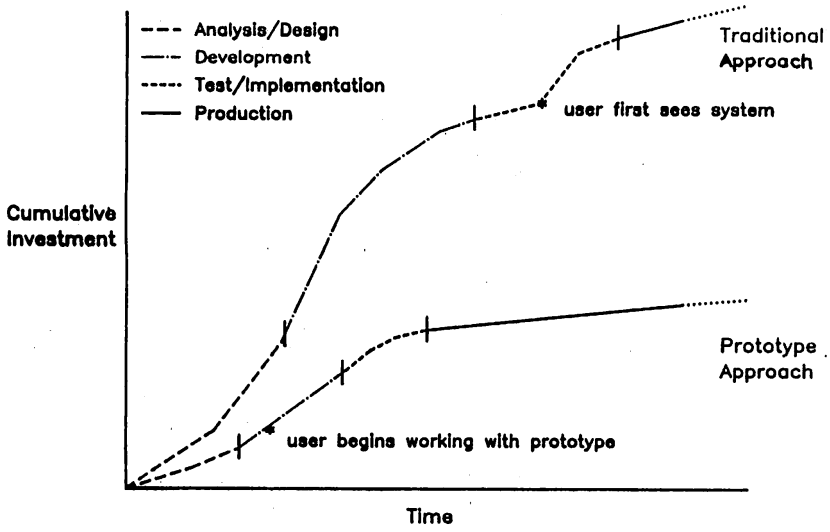
- Programmer/analyst demonstrates prototype to small group of users.
- Users may request enhancements during demo.
- Users make notes of all changes they would like made.

Step 4. Revise and enhance the prototype:

- Programmer/Analyst and user discuss desired changes.
- Changes and enhancements for the next version are prioritized.
- Programmer/Analyst creates next version.
- Go back to Step 3.

NOTE: Steps 3 and 4 are repeated until the system achieves the requirements of this small group of users. Then either introduce to a larger group of users for additional requirements or if enough users are satisfied, demo to management to gain approval for the production system.

PROTOTYPING VS TRADITIONAL APPROACH



Uses of Software Prototypes

1. To clarify user requirements:

   - Written specs are often incomplete, confusing, and take a static view of requirements.
   - It is difficult for an end user to visualize the eventual system, or to describe their current requirements.
   - It is easier to evaluate a prototype than written specifications.
   - Prototyping allows - even encourages users to change their minds.
   - It shortens the development cycle and eliminates most design errors.
   - It results in less enhancement maintenance and can be used to test out the effects of future changes and enhancements.

2. To verify the feasibility of design:

   - The performance of the application can be determined more easily.
   - The prototype can be used to verify results of a production system.
   - The prototype can be created on a minicomputer and then that software prototype may become the specifications for that application which may be developed on a larger mainframe computer.

3. To create a final system:

   - Part (or all) of the final version of the prototype may become the production version.
   - It is easier to make enhancements and some parts may be recoded in another language to improve efficiency or functionality.

Essential Resources

The following are the essential resources to effectively do software proto- typing:

1. Interactive Systems

   - Hardware and Operating System - When doing software prototyping, both the builder and the system must respond rapidly to the user's needs.

Batch systems do not permit interaction and revision at a human pace. Hardware and associated operating systems tailored to on-line interac- tive development are ideal for software prototyping.

2. Data Management Systems

- A Data Base Management System provides the tools for defining, creating, retrieving, manipulating, and controlling the information resources. Prototyping without a DBMS is inconceivable!

- A Data Dictionary provides standardization of data and file locations and definitions, a cross reference of application programs, and a built- in documentation capability. These are essential to managing the corporate resources and extremely useful when prototyping.

3. Generalized Input and Output Software

- Easy to use data entry, data editing, and screen formatting software are extremely helpful in the software prototyping process to allow the programmer to sit down at a terminal with a user and interactively create the user's screens or menus.

- Powerful easy-to-use report writer and query languages provide a quick and effective way of retrieving and reporting on data in the system. A report writer that uses default formats from very brief specifications is most useful in the initial prototype.

4. Very High Level Languages

- Traditional application development languages such as COBOL may not be well suited for software prototyping because of the amount of code that has to be written before the user sees any results.

- Very powerful high level (MACRO) languages that interface directly to a data dictionary for their data definitions are ideal. One statement in this high level language could realistically replace 20-50 COBOL state- ments. This reduces the amount of code a programmer has to write and maintain and speeds up the development process.

5. Library of Reusable Code

- A library of reusable code to reduce the amount of redundant code a programmer has to write is an important prototyping resource.

- This code could represent commonly used routines made available to programmers.

Potential Problems

What are the problems with prototyping? How can data processing management control its use and keep it within bounds?

One problem with prototyping is the acceptance of this method by the systems people. It also may encourage the glossing over of the systems analysis portion of a project. It may be difficult to plan the resources to develop a system. Programmers may become bored after the nth iteration of the proto- type. Testing may not be as thorough as desired and it might be difficult to keep documentation on the application up to date because it is so easy to change.

Even with these concerns, prototyping provides a very productive user-designer working relationship. So it behooves all data processing executives to learn to use this powerful tool creatively and to manage it effectively.

The advantages of prototyping greatly outweigh the problems.

Cost and Efficiency

It has been found that there is an order of magnitude decrease in both develop- ment cost and time with the prototype model.

It is often difficult to estimate the cost of an application system because the total costs of development, including maintenance are usually lumped together. The cost of implementing the initial system is much lower than the traditional approach (typically less than 25%).

However, software prototyping could be expensive in three ways:

1. It requires the use of advanced hardware and software.
2. It requires the time of high level users and experienced designers.
3. Efficiency may be compromised.

The main thing to remember is that the main focus of prototyping is not so much efficiency but effectiveness.

Summary

Prototyping is truly a "state of the art" way of developing applications.

- Software prototyping promotes an interactive dialogue between the users and the programmer, which results in a system being developed more quickly, and results in an interactive development approach which is friendlier for the end user.

- The prototype provides a live working system for the users to experiment with instead of looking at lengthy specifications.

- The users are provided with an early visualization of the system which allows them to immediately use it.

- The users are allowed and even encouraged to change their minds about user interfaces and reports.

- Maintenance is viewed right from the beginning as a continuous process and because the prototype is usually written in a very high level lan- guage, changes are faster to locate and easier to make.

- Software prototyping results in:

* Users who are much more satisfied and involved in the development process.

* Systems that meet the user's requirements and are much more effective and useful.

* Improved productivity for all those involved in software prototyping: the users, the analysts, and the programmers.

Hewlett-Packard's Prototyping Tools

Hewlett-Packard is one of the few vendors that supplies the majority of the tools needed to effectively do software prototyping.

* Interactive Systems

- HP3000 (All Series) - MPE Operating System

* Data Management Systems

- IMAGE/3000 - KSAM/3000 - MPE files - DICTIONARY/3000

* Generalized Input/Output Software

- VPLUS/3000 - QUERY/3000 - REPORT/3000 - INFORM/3000 - DSG/3000

* Very High Level Languages

- TRANSACT/3000

Canning, Richard G., "Developing Systems By Prototyping," EDP Analyzer (19:9) Canning Publications, Inc., September, 1981.

Naumann, Justus D. and Jenkins, A. Milton, "Prototyping: The New Paradigm for Systems Development," MIS Quarterly, Vol. 6, No. 3, September 1982.

Naumann, Justus D., and Galletta, Dennis F., "Annotated Bibliography of Proto- typing for Information Systems Development," Management Information Systems Research Center Working Paper (MISRC-WP-82-12), September 1982.

Note: The above working paper as well as the paper by Naumann and Jenkins entitled "Prototyping: The New Paradigm for Systems Development," MIS Research Center-Working Paper (MISRC-WP-82-03), October 1981, are available for $3.00 each from :

University of Minnesota Systems Research Center School of Management 269 19th Avenue South University of Minnesota Minneapolis, Minnesota 55455

or by calling 612-373-7822.

Podolsky, Joseph L., "Horace Builds a Cycle," Datamation, November 1977, pp.162-186.