INFORMATION SYSTEMS PROTOTYPING

Orland Larson
Hewlett-Packard Company, Cupertino,California, USA

Summary

One of the most imaginative and successful techniques for clarifying user interfaces and generally improving the productivity and effectiveness of application development is a methodology called INFORMATION SYSTEMS PROTOTYPING.

With waiting time for new applications running into several years and those applications failing to meet the users needs, managers as well as users have been searching for more efficient and effective approaches to systems development.

Prototyping, as an application system design and development methodology, has evolved into a real option for both the MIS professional and the user.

This paper reports on the growing body of knowledge about prototyping. It begins by reviewing the changing role of data processing, the challenges facing the MIS organization, and the traditional approach to application development. It then defines prototyping followed by the step-by-step prototype development process. The advantages and disadvantages, as well as the cost and efficiency of prototyping, will be discussed followed by the essential resources neccessary to effectively prototype applications. In conclusion, to illustrate the benefits of prototyping, the speaker will present success stories of systems developed using the prototyping approach.

INTRODUCTION

The Changing Role of Data Processing

The data processing department has changed dramatically since the 1960s, 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development as well as production jobs were usually run in a batch environment with long turnaround times and out-of-date results.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 1970s 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 programs on-line, while 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 1980s, the data base administrator and MIS manager will see increased importance and use of centralized data dictionaries or "centralized repositories of information about the corporate data resources." Simpler and more powerful report writers will be used by the end user and business professional. The programmer will see the trend towards the use of high-level, transaction processing languages, also known as fourth generation languages, to reduce the amount of code required to develop applications. Finally, the tools have been developed to effectively do application prototyping, which will provide benefits to the end user as well as the application programmer and analyst.

Throughout the 70s and 80s, information has become more accurate, reliable, and available, and the end user or business professional is becoming more actively involved in the application development process.

Challenges Facing MIS

One of the MIS manager's major problems 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 (typically, 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 to 80 percent of their time). Much of the code the programmer generates includes 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 counterproductive 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 users' 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 a while. He is frustrated when he requests what he thinks is a "simple change" and MIS takes weeks or months to make that change.

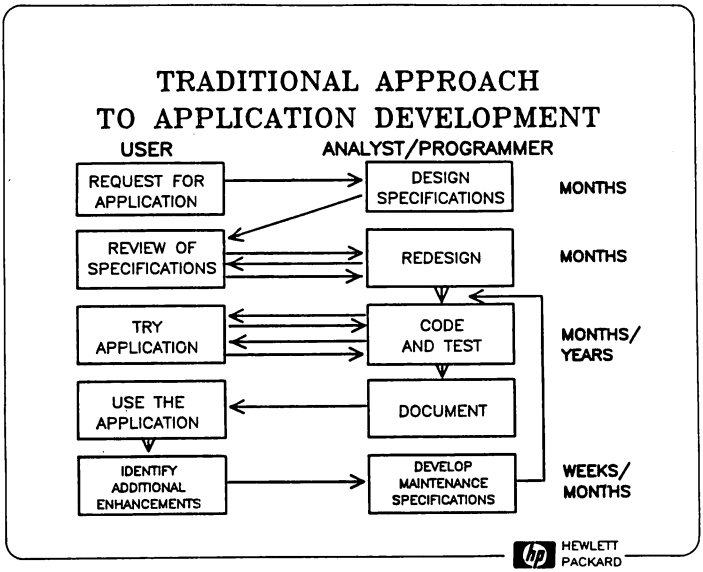
Traditional Approach to Application Development

There are some myths concerning traditional application development:

- Users know exactly what they want
- Users can effectively communicate their needs to MIS
- Users needs never change.

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.



- 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's 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 finally approves the specifications, and a team of analysts and programmers are assigned to develop, test and document the application.

- 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 user, of course, will most likely want additional changes or enhancements made to the application. This is called adjusting the application to the "real world".

- Depending on the extent of these changes, additional maintenance specifications may have to be written and these program changes coded, tested and documented.

- The total application development process may take months or years, and the maintenance of these applications may go on forever.

In summary, the traditional approach to application development results in long development times, excessive time spent on maintenance, a multi-year backlog of applications, limited control and access to information, and applications that lack functionality and flexibility and are very difficult to change. The question is: "Can we afford to continue using this approach to application development?"

Prototype 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 7) 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.

Roles in the Prototyping Process

There are two roles to be filled in prototyping -- the user/designer and the systems/builder. These roles are very different from the traditional user and analyst/programmer roles under the traditional approach. The terms "user/designer" and "systems/builder" emphasize these differences and denote the functions of each participant under the prototyping methodology. Remember it is the user who is the designer of the application system and the systems professional who is the builder.

The user/designer initiates the process when he/she conceives of a problem or opportunity that may be solved or exploited by the use of an information system. The user/designer typically must be competent in his/her functional area (many times he/she is a manager) and usually has an overall perspective of the problem and can choose among alternative solutions. However, he/she requires assistance from the MIS organization.

The systems/builder is assigned by the MIS organization to work with the user/designer and is competent in the use of the available prototyping tools and knowledgeable about the organizations data resources.

Prototyping Process

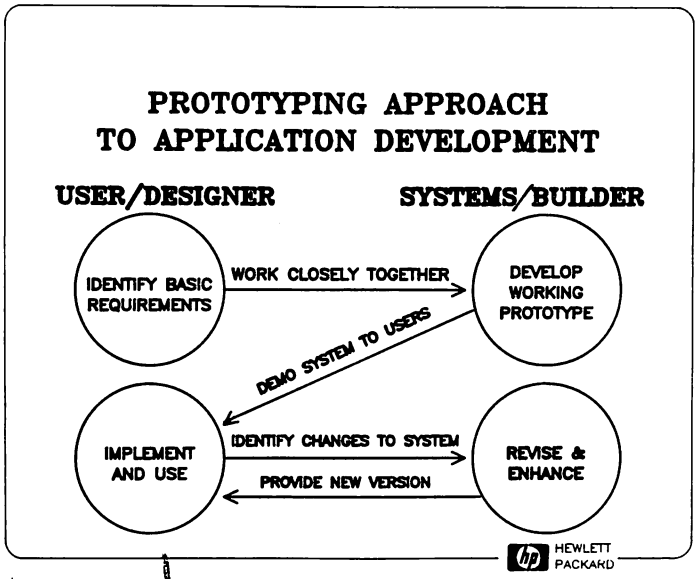
The process of application prototyping is a quick and relatively inexpensive process of developing and testing an application system. It involves the user/designer and the systems/builder working closely to develop the application. It is a live, working system; it is not just an idea on paper. It performs actual work; it does not just simulate that work. It can be used to test 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.

Prototyping Methodology/Model

The prototyping methodology in general, is based on the following proposition: "People can tell you what they don't like about an existing application easier than they can tell you what they think they would like in a future application."

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



Step 1. User/designer identifies the basic information requirements:

- Write a brief, skeleton-like statement that captures the essential features of the information requirements.
- User/designer and systems/builder work closely together.
- Concentrate on users' most basic and essential requirements.
- Define data requirements, report formats, screens, and menus.
- Need not involve lengthy written specifications.
- For larger systems, a design team may need to spend a few weeks preparing a first-effort requirements document.

Step 2. Systems/builder develops the initial prototype:

- Systems/builder takes the notes developed in the user discussions and quickly builds the menus and dialogs.
- A data dictionary would be useful at this time.
- Design and/or define data base and load subset of data.
- Make use of defaults and standard report formats.
- Write required application modules using a fourth generation language.
- Prototype performs only the most important, identified functions.

Step 3. Users implement and use the prototype to refine requirements:

- Systems/builder demonstrates prototype to small group of users.
- Users gain hands-on experience with application.
- Users are encouraged to make notes of changes they would like made.
- Users discuss and prioritize desired changes.

Step 4. Systems/builder revises and enhances the prototype:

- Systems/builder modifies the prototype to correct undesirable or missing features.
- May require modification or redesign of data base, changes to existing programs and/or additional program modules.
- Deliver back to users quickly.

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

When to Use Prototyping

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 his/her 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 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.

When Not to Use Prototyping

1. When an application requires a standard solution that already exists and is available at a reasonable cost from a software supplier.

2. When you don't have a good understanding of the tools available to prototype.

3. When the organization's data and software resources are not well organized and managed.

4. When MIS management is unwilling to develop a staff of professional systems/builders.

5. When the user/designer is unwilling to invest his/her time in the development of the application system.

Potential Problems

One of the initial problems typically encountered is the acceptance of the prototyping methodology by the systems people. This is due to the fact that people naturally tend to resist change. It may also encourage the glossing over of the systems analysis portion of a project. It is not always clear how a large complex system can be divided and then integrated. Initially, it could be difficult to plan the resources required to prototype (people, hardware and software). It may be difficult to keep the systems staff and users abreast of each version of the system. Programmers may tend to become bored after the nth iteration of the prototype. Testing may not be as thorough as desired. 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 working relationship for the users and the builders. So it behooves all data processing management to learn to use this powerful tool creatively and to manage it effectively.

THE ADVANTAGES OF PROTOTYPING GREATLY OUTWEIGH THE PROBLEMS!

Advantages of Prototyping

One of the main advantages of application prototyping is that this methodology provides a capability to quickly respond to a wide variety of user requests. It provides a live, functioning system for user experimentation and accommodates changes in a dynamic user environment. One interesting aspect of this approach is that users are allowed and even encouraged to change their minds about an application's interfaces and reports, which is a very rare occurrence during the traditional approach. Maintenance is viewed right from the beginning as a continuation of the design process. Finally, prototyping provides an effective use of scarce systems/builders. One or a limited number of systems/builders will be required for each prototyping project; and while users are testing one prototype, the systems/builder can be working on another.

Cost and Efficiency

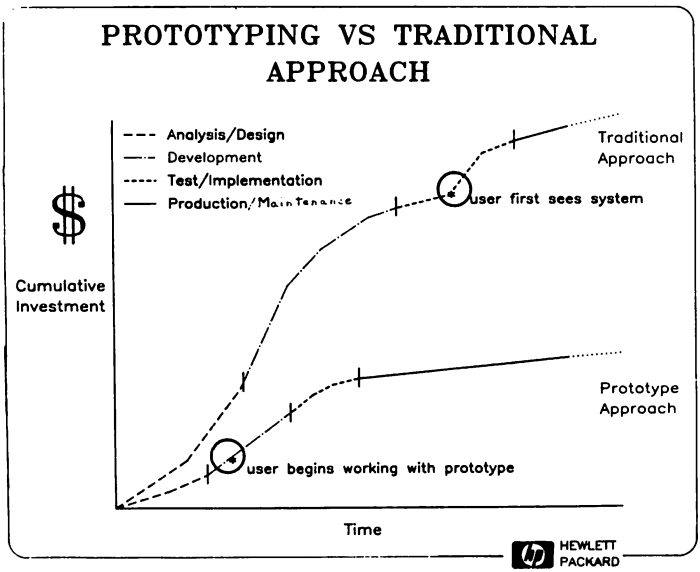
It has been found that there is an order of magnitude decrease in both development cost and time with the prototyping methodology.

It is often difficult to estimate the cost of prototyping 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 the following ways:

- It requires the use of advanced hardware and software.
- It requires the time of high-level users and experienced systems staff.
- It requires training of the systems staff in the use of prototyping and the associated tools.
- Application run-time efficiency may be compromised.

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



Essential Resources

The following are the essential resources to effectively do application prototyping:

1. Interactive Systems

   - Hardware and Operating System - When doing application 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 interactive 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!

3. Data Dictionary

   - 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.

4. Generalized Input and Output Software

   - Easy to use data entry, data editing, and screen formatting software are extremely helpful in the application 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.

   - A powerful graphics capability can be extremely useful for the display of data in a more meaningful graphical format.

5. Very High Level (Fourth Generation) 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 fourth generation 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 statements. This reduces the amount of code a programmer has to write and maintain and speeds up the development process.

6. Documentation Aids

   - Tools to aid in the maintenance of programs written in a 4GL.

   - Tools to aid in maintaining user documentation on-line.

7. Libraries of Reuseable 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.

Hewlett-Packard's Tools for Prototyping

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

* Interactive Systems

  - HP 3000 Family of Computers
  - MPE Operating System

* Data Management Systems

  - IMAGE/3000
  - KSAM/3000
  - MPE files
  - HP Silhouette/3000
  - HP Access Central, HP Access

* Data Dictionary

  - Dictionary/3000

* Generalized Input/Output Software

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

* Very High Level Languages

  - TRANSACT/3000

* Documentation Aids

  - EDITOR/3000
  - HPSLATE
  - HPWORD
  - TDP/3000

Additional Prototyping Tools Available from HP Third-Party Vendors

* Data Management Systems and Associated Utilities

- ADAGER	Adager
- CARESS, INTACT, SILHOUETTE/3000	Carolian Systems International, Inc
- DBACE	Snodgrass Consulting
- DBAUDIT, SUPRTOOL	Robelle Consulting Ltd.
- DB GENERAL	Bradmark Computer Systems
- DBMGR, IMSAM, OMNIDEX, CAPCHG	Dynamic Information Systems, Corp.
- DBTUNE (Europe Only)	HI-COMP
- HSC-COPYDB	Hawaiian Software Company
- IMAGINE	Technalysis Corporation
- MINISIS	Systemhouse Ltd.
- MIRAGE (HP 150)	Datasoft International
- PC/IMAGE (HP 150)	Advanced Data Services
- RELATE/3000	CRI, INC.
- SPEEDEX, SPEEDBASE (HP 150)	Infocentre

* Generalized Input/Output Software

- DATADEX/3000	Dynamic Information Systems, Corp.
- EASYREPORTER	Infocentre
- ENVY, HELPER	System Works, Inc.
- INDEX PLUS	Spectrum Solutions
- MONITOR, MISTRAL (HP 150)	Datasoft International
- PAL DATA REPORTER	Gentry
- PRESENTATION GRAPHICS	ARENS
- PRW/3000	Infotek Systems
- QUIZ, THE EXPERT, GRAPHICS	COGNOS
- RELATIONAL QUERY/3000	Upland Software
- SCREEN/3000	RMS Business Systems
- WHAT-IF	CIBAR, Inc.
- THE WRITE STUFF	PROTOS Software Company

* Fourth Generation Languages and Utilities

- ARTESSA/3000 (Europe only)	RAET Software Products
- CBAS/3000	Comprehensive Systems, Inc.
- FASTRAN (TRANSACT Compiler)	Performance Software Group
- FLEXIBLE	Sages American Group
- INSIGHT II	Computing Capabilities Corp.
- LL'SPIRIT	Singapore Computer Systems, PTE.LTD.
- PAL FAMILY	GENTRY
- POWERHOUSE (QUICK)	COGNOS
- PROGSPEC/3000 (COBOL Gen.)	Productive Systems
- PROTOS (COBOL Generator)	PROTOS Software Company
- Q-PLUS	Los Altos Software
- RELATE/3000 APPLICATION BUILDER	CRI
- SPEEDWARE, MICROSPEEDWARE	Infocentre
- THE SYNERGIST	Gateway Systems Corp.

* Documentation Aids

- DOCUMENTOR (Part of SPEEDWARE)	Infocentre
- LARC	LARC Computing
- QEDIT	Robelle Consulting Ltd.
- ROBOT/3000	Productive Software Systems, Inc.
- S/COMPARE	Aldon Computer Group
- SPEEDDOC, SPEEDEDIT	Bradford Business Systems, Inc.
- TESS/AIDE	Computer Consultants and Serv. Center

The preceding lists of HP third-party software are not 100% complete. The majority of the listed software was derived from ads placed in SuperGroup Association Magazine, Interact Magazine and The Chronicle. Please consult the Hewlett-Packard Business Systems Software Solutions catalog (Part # 30000-90251) for additional information.

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 language, 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 user/designers and the systems/builders.

Biography

Orland Larson
is currently Information Resource Management Specialist for Hewlett-Packard. As the data base and application development specialist for the Information Systems Tactical Marketing Center he develops and presents seminars worldwide on data base management, information systems prototyping and productivity tools for information resource management. He is a regular speaker at Hewlett-Packard's Productivity Shows and Users Group meetings and also participates in various National Data Base and 4th Generation Language Symposiums. Previously he was the Product Manager for IMAGE/3000, Hewlett-Packard's award winning data base management system.

Before joining HP he worked as a Senior Analyst in the MIS Department of a large California-based insurance company and prior to that as a Programmer/Analyst for various software companies. Mr. Larson has been with Hewlett-Packard since 1972.

Bibliography

Boar, Bernard H., Application Prototyping: A Requirements Definition For The 80's, John Wiley & Sons, New York, New York, 1984.

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

Jenkins, A. Milton, "Prototyping: A Methodology For The Design and Development of Application Systems," Division of Research, School of Business, Indiana University Discussion Paper #227, April 1983, (41 pages).

Jenkins, A. Milton and Lauer, W. Thomas, "An Annotated Bibliography on Prototyping," Division of Research, School of Business, Indiana University Discussion Paper #228, April 1983, (25 pages).

Larson, Orland J.,"Software Prototyping - Today's Approach to Application Systems Design and Development," Proceedings 1984 International Meeting HP 3000 IUG, Anaheim, California, February 26 - March 2.

Martin, James, Application Development Without Programmers, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1982.

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 Prototyping for Information Systems Development," Management Information Systems Research Center Working Paper (MISRC-WP-82-12), September 1982.

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

Wetherbe, James C., "Systems Development: Heuristic or Prototyping," Computerworld, Vol. 16, No. 7, April 26, 1982.