Designation: E 1340 – 96

An American National Standard

# Standard Guide for Rapid Prototyping of Computerized Systems[1]

This standard is issued under the fixed designation E 1340; the number immediately following the designation indicates the year of original adoption or, in the case of revision, the year of last revision. A number in parentheses indicates the year of last reapproval. A superscript epsilon (ε) indicates an editorial change since the last revision or reapproval.

## 1. Scope

1.1 This guide covers a rapid prototyping method for developing computerized systems. Intended readers of this guide are people who develop computerized systems, and students and teachers of system development methods.

1.2 Rapid prototyping is an approach to developing computerized systems which produces a working model more quickly than conventional approaches. Where conventional methods concentrate on preparing functional requirements and functional design documents that describe the needed system, rapid prototyping methods concentrate on preparing a working prototype. Users and developers learn the functional requirements and an appropriate system design by interacting with a series of prototypes, each of which is rapidly produced from a starting framework or from an earlier version. A prototype can evolve into an operational system, it can serve as an exact behavioral specification of an operational system, or it can be used to explore the feasibility of a new idea or design which can be incorporated in a larger system. The method is rapid in preparing each version of the prototype, but the overall time required for system development may be more or less than the time required with conventional methods.

1.3 Rapid prototyping is most appropriate when the functional requirements or functional design for a system are not well understood, or when experimentation is required to explore some aspect of system behavior. It is not appropriate in hazardous settings, or when the requirements are well understood.

1.4 The guide recommends use of prototyping tools, but it is not a standard for the tools themselves. It does not cover executable specification tools. Transforming a prototype that is used to clarify requirements into an operational system is discussed briefly in Section 8 and in detail in other referenced standards (see 2.1).

1.5 *This standard does not purport to address all of the safety concerns, if any, associated with its use. It is the responsibility of the user of this standard to establish appropriate safety and health practices and determine the applicability of regulatory limitations prior to use.*

## 2. Referenced Documents

2.1 *ASTM Standards:*

E 622 Guide for Developing Computerized Systems[2]

E 625 Guide for Training Users of Computerized Systems[2]

E 627 Guide for Documenting Computerized Systems[2]

E 731 Guide for the Selection and Acquisition of Commercially Available Computerized Systems[2]

E 919 Specification for Software Documentation for a Computerized System[2]

E 1013 Terminology Relating to Computerized Systems[2]

E 1029 Guide for Documentation of Clinical Laboratory Computer Systems[2]

2.2 *ANSI Standards:*

ANSI/MIL-STD-1815A Ada Programming Language[3]

ANSI/X3.9 Programming Language FORTRAN[3]

ANSI/X3.159 Programming Language C[3]

ANSI/X11.1 MUMPS Programming Language[3]

ANSI/IEEE 729 Glossary of Software Engineering Terminology[3]

ANSI/IEEE 770 X3.97 Pascal Programming Language[3]

ANSI/IEEE 1063 User Documentation for Computer Software[4]

## 3. Terminology

3.1 *Definitions*—For definitions of terms relating to computerized systems, refer to Terminology E 1013, IEEE 729, and ANDIP.[5]

3.1.1 *fourth generation language*, *n*—a high-level computer language that incorporates data structures and procedures for a specific problem domain.

3.1.2 *prototype*, *n*—an original or model from which a system is copied.

3.1.3 *prototype*, *v*—to create an original or model.

[1] This guide is under the jurisdiction of ASTM Committee E31 on Healthcare Informatics and is the direct responsibility of Subcommittee E31.23 on Modeling for Health Informatics.

Current edition approved Oct. 10, 1996. Published December 1996. Originally published as E 1340 – 90. Last previous edition E 1340 – 91.

[2] *Annual Book of ASTM Standards*, Vol 14.01.

[3] Available from American National Standards Institute, 11 W. 42nd St., 13th Floor, New York, NY 10036.

[4] Available from the Institute of Electrical and Electronics Engineers, Inc., 345 E. 47th Street, New York, NY 10017.

[5] American National Dictionary for Information Processing Systems, Information Processing Systems Technical Report X3/TR-1-82, Dow Jones-Irwin, Homewood, IL.

3.1.4 *prototyping*, *n*—the activities that create an original or model.

3.1.5 *rapid prototyping*, *n*—an iterative method for developing prototypes of components, subsystems, or complete computerized systems, in which the time between successive versions of the prototype is short.

3.1.6 *RP*, *n*—rapid prototyping.

3.1.7 *third generation language*, *n*—a procedural high-level computer language, such as COBOL, FORTRAN, or Pascal.

## 4. Significance and Use

4.1 Rapid Prototyping (RP) is a way to develop a computerized system which produces a working model of the system very quickly. The RP process shown in Fig. 1 has many similarities, and some differences from the conventional system development process shown in Fig. 2. RP replaces the functional requirements and functional design phases of the conventional method with an iterative process of prototype refinement. Where the phases of the conventional method produce a set of documents that describe the system, RP produces a prototype. The prototype is tested and refined through several iterations, with intense interaction between system users and developers. RP is an experimental approach to system development which provides a learning device, the prototype, for users and developers. A prototype can be used as a tool for clarifying functional requirements for the operational system, as a means of evaluating a design approach, or as a developing series of versions of the operational system. A prototype is sometimes used as an exact behavioral specification for an operational system which replaces it. Quality characteristics are often sacrificed during RP for the sake of rapid development and low cost; robustness, efficiency, generality, portability, and maintainability are commonly ignored. However, documentation needed to use the system cannot be ignored.

4.1.1 *Rapid* in RP means that the time between successive versions of the prototype is short. It should be short enough that (*1*) both users and developers can remember how each version relates to the previous one without written notes, (*2*) user requirements do not change significantly while a version is being developed, (*3*) the prototyping team will remain in the project through the RP phase, and ( *4*) total time to develop the system is acceptable. (Expected project duration should be stated in the project definition agreement. See Section 6 and Guide E 622, Section 6.) A few days between versions is adequate and a few weeks may be acceptable. If the time needed to produce a new version is longer, then it may be necessary to produce that version using a conventional system development method (for example, Guide E 622) with full documentation of requirements and design (see Appendix X3).

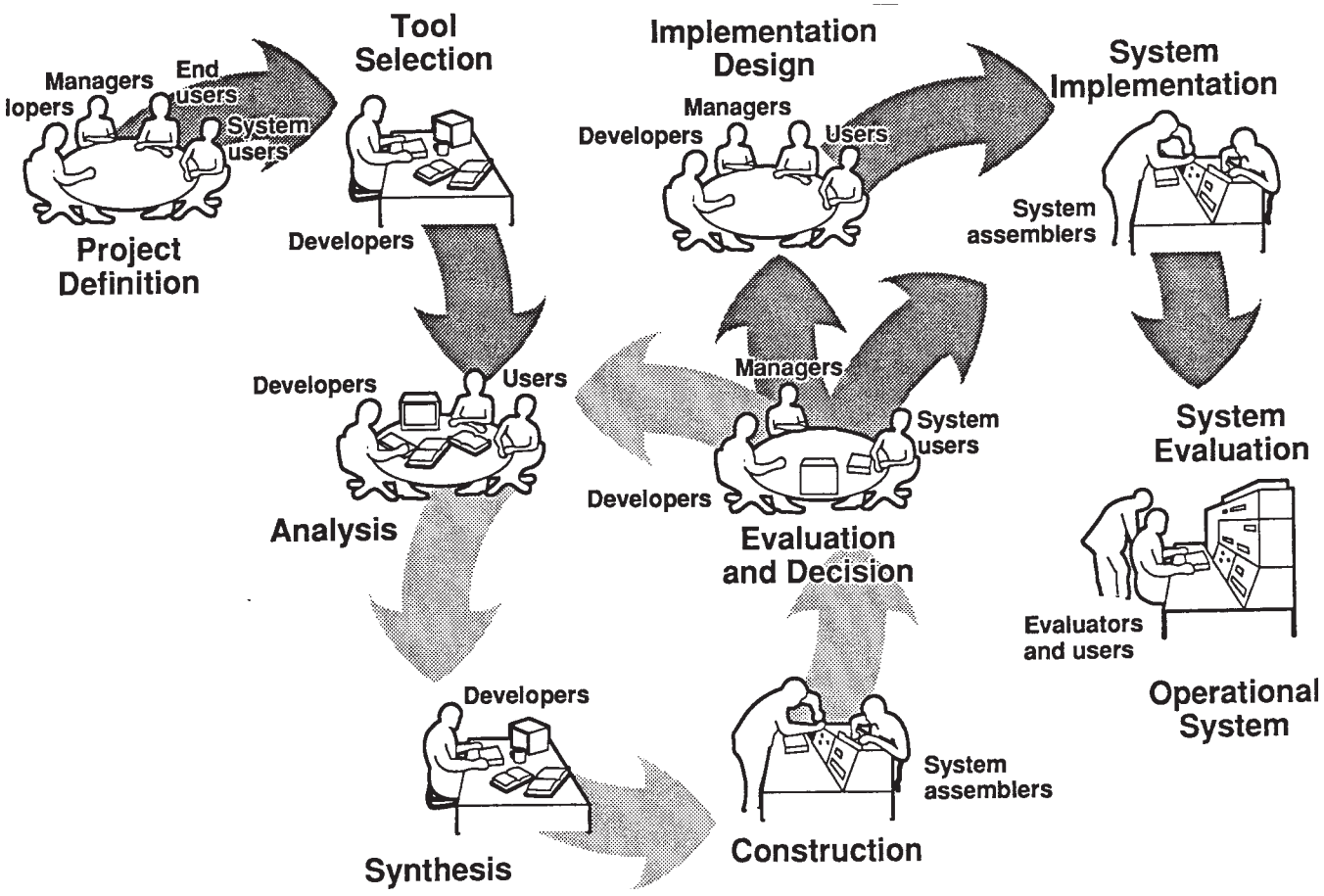


**FIG. 1 Rapid Prototyping of a Computerized System**

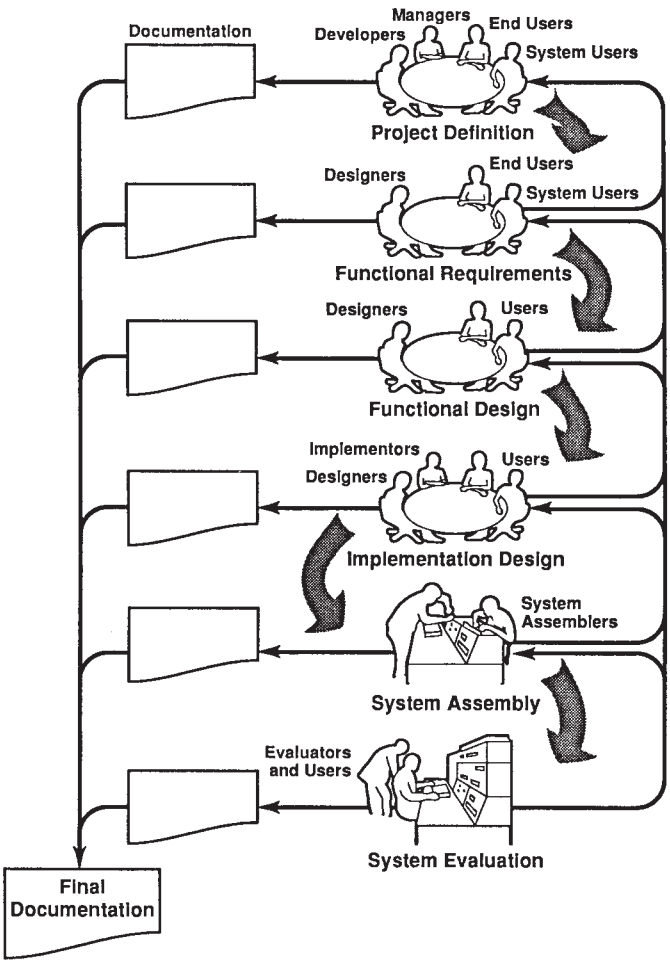


**FIG. 2 Conventional Development of a Computerized System**

4.1.2 RP integrates analysis, design and construction, and defines requirements during the process. It is especially appropriate for dealing with problems which are not well understood. The prototype focuses communication between users and developers.

4.2 For large systems, a RP approach can be used at a high level to explore the overall system architecture or feasibility. It can also be used to develop subsystems and components whose requirements are not fully understood (see Section 11). RP is especially well suited for developing user-system interfaces.

4.2.1 *What to Prototype*—The ill-structured system development problems that yield best to RP include:

4.2.1.1 Decision support systems in areas where the state of knowledge changes rapidly, for example, research or clinical practice,

4.2.1.2 Systems whose users need to access and organize data in ways not foreseen when the system is created, for example, strategic decision support,

4.2.1.3 Systems that consist entirely of software,

4.2.1.4 Instructional or experimental systems, and

4.2.1.5 User-system interfaces.

4.2.2 *Ways to use RP*—Three ways that are widely used are (*1*) evolutionary, ( *2*) experimental, and (*3*) build-and-replace. In evolutionary prototyping, the developers rapidly produce an initial version as a framework to learn user requirements, and then satisfy the requirements incrementally through a series of versions to produce the operational system. In experimental prototyping, the developers explore the feasibility of selected capabilities or components with a series of versions that serve to test concepts and designs. In build-and-replace prototyping, the developers assemble a series of versions to establish what the system should do and how it should do it, then the prototype is used as a behavioral specification for building the operational system. Build-and-replace is sometimes called throw-away prototyping, but the prototype should not be thrown away.

## A RAPID PROTOTYPING METHOD

### 5. Introduction

5.1 The following sections describe a system development method that uses RP. It is based on, and shares some features with, the method described in Guide E 622 and other ASTM standards which are listed in 2.1. Instead of producing documents that describe functional requirements (Guide E 622, Section 7), functional design (Guide E 622, Section 8), and implementation design (Guide E 622, Section 9) for the required system, this method produces a prototype of the system and refines it through an iterative process of analysis, synthesis, and evaluation.

### 6. Project Definition

6.1 In any system development project, whether done by RP or conventional means, it is important to have a definition of what is to be accomplished, when, where, why, by whom, and for about how much effort. It is also important to identify everyone who must be satisfied with the results, and especially everyone who will use the system. These things are determined in the preliminary discussions and negotiations about a project. A written statement of them is a project definition document. Guide E 622, Section 6 says that a project definition is an agreement among everyone involved with a project. Agreement on project goals is essential for project success.

6.2 A project definition should be in writing. A written document is concrete and changes to it are explicit. A written project definition does not drift like a verbal one. As new people become involved with the project, they can read the original document and can either become parties to it or renegotiate it.

6.3 The project definition document should be brief, preferably no more than one page. If it exceeds two pages, provide a one page summary.

6.4 The project definition document should state what is to be prototyped and it should be specific about the goals of the project. If a prototype is to be developed to learn the true requirements for a system, the project definition should say so. If a prototype is to explore feasibility of a new idea or design, that should be stated in the project definition. The project definition document should say whether the prototype is to evolve into an operational version, or is to be replaced by an operational version which is to be rebuilt from scratch. If this is not known at the start of the project, then the project definition should state the criteria for deciding.

6.5 The project definition document should briefly say what functions the prototype is to perform. It should clarify and limit the scope of the project, the prototype, and the system that is to be based on the prototype.

6.6 Everyone involved with the project should indicate agreement by signing the project definition document, or the one page summary of it.

## 7. Tool Selection

7.1 The project definition and users' preliminary statements about what the system needs to do, guide the developers in selecting appropriate RP tools. The initial choice is crucial because it immediately constrains what can be accomplished. By selecting the right tools, the developers minimize the time and effort required for each RP cycle and maximize the likelihood of success. If the wrong tools are selected, then the project may be immediately doomed to failure. Most of the tools discussed in this section are software tools, that is, computer programs, but the principles also apply to hardware tools.

7.2 Good tools for RP share the following characteristics:

7.2.1 They help produce the prototype quickly,

7.2.2 They allow changes to the prototype to be made quickly,

7.2.3 They are general enough to permit more than one solution to most problems,

7.2.4 They encourage developers to try different approaches,

7.2.5 They are used interactively,

7.2.6 The person using them need not be a highly trained specialist,

7.2.7 They simulate real time, and

7.2.8 They are available for use now rather than proposed or promised for future use.

7.3 *RP languages*:

7.3.1 Many fourth generation languages are appropriate for RP. A good RP language uses the natural terminology of the problem domain, and it incorporates enough knowledge about programming to eliminate most of the need to write procedures. A good RP language should incorporate concepts which are in general use by the people who are to use it. It should provide high-level data constructs that are appropriate to the problem.

7.3.2 The user-system interface for a good RP language should incorporate features which allow development team members to use descriptive techniques that they use with human colleagues, that is, graphic, tabular, and other nonverbal techniques.

7.3.3 Tools that make the job easy for an inexperienced person may be tedious for a skilled system engineer. There should be a terse dialog mode for the experienced user who can abstract more features than the inexperienced user.

7.3.4 Good languages for RP are usually directly executed or interpreted in some fashion. The ideal language for an evolutionary prototype would fit the problem domain, would have an interpretive processor with good diagnostic facilities, and would be compilable (for efficiency of the operational version).

7.3.5 If the prototype is to evolve into the operational system, then the RP language should have reasonable efficiency. MUMPS (ANSI X11.1), LISP, and APL are widely used RP languages. These languages make RP easier than strongly typed languages like Pascal (ANSI/IEEE 770 X3.97) and Ada (ANSI/MIL-STD 1815A).

7.3.6 General purpose third generation programming languages like FORTRAN (ANSI X3.9) and C (ANSI X3.159) are not appropriate for RP unless powerful libraries can be used to eliminate the bulk of the programming effort. These languages may be useful if the prototype is to evolve into an operational system, but they are not appropriate for RP unless a large body of code is available and well-suited for reuse in the prototype. These languages may also be useful in an environment where the prototype can use several languages, and language selection is less critical.

7.4 RP is especially useful for developing user-system interfaces, even when other system functions are developed by a conventional approach. Mock-ups and simulators are powerful tools for exploring how people will use a system to accomplish their tasks. The interface between a system and its users can determine whether the system is accepted and successful. Tools for prototyping user-system interfaces should allow system developers to describe dialog attributes and style in plain language. These kinds of tools are useful for RP of user-system interfaces:

7.4.1 Screen managers, screen layout simulators, and forms managers, to rapidly produce user displays. These should incorporate graphics which are appropriate for the problem domain.

7.4.2 Dialog generator such as a transition diagram interpreter. This allows RP of a dialog that involves several different user displays.[6]

7.4.3 Dialog recording, to capture the conversation between user and system for later playback and analysis.

7.4.4 Report generator to rapidly produce printed output.

7.5 *There are two kinds of databases in RP*—(*1*) Data used by the system itself. Development of this database may be one of the goals of a RP project. ( *2*) Data about the project including past, present and future states of the system design, that is, the history, current version and plans for the prototype. Good tools make effective management of both kinds of databases easier. Capabilities for data updates, retrieval, security, backup, and transaction logging may be required in database management tools. The database management tools should allow data structures that are natural to the problem domain. Data should not be forced into inappropriate structures because of inappropriate tools.

7.6 *Prefabricated components*, both hardware and software, can minimize the effort to produce a prototype. Sets of reusable modules which are available off the shelf can minimize software development effort like a good electronic parts catalog minimizes hardware development effort. It is preferable to build a system out of large hardware and software

[6] Wasserman, A. I., and Shewmake, T., "Rapid Prototyping of Interactive Information Systems,'' Working Paper from ACM SIGSOFT Workshop on Rapid Prototyping. *ACM Software Engineering Notes,* Vol 7, No. 5, December 1982, pp. 171–180.

components in order to minimize the effort to design, construct, and test each prototype version.

7.6.1 Reusable software is especially useful, tools like subroutine libraries and packages of data types and functions. Analysis, design, and tested code should be reused. Reuse of components requires careful specification of interfaces.

7.6.2 Skeleton programs, which require only the details to be filled in, can reduce coding and debugging.

7.7 *Software generating tools*:

7.7.1 Some software systems (for example, table-driven code generators) allow different members of a family of programs to be created by varying parameters or tables. Each member is an instance of a more general program.

7.7.2 Application or program generators, or automatic programming systems are available for a few problem domains.

7.7.3 Computer-aided software engineering (CASE) tools that can generate source code may be useful for RP. CASE tools that require manual translation of the software design into code may be useful in conventional system development which focuses on design documents, but they are not appropriate for RP.

7.8 *Other useful tools for RP*:

7.8.1 Statistical packages with graphics,

7.8.2 Test harnesses, both software and hardware, to simulate the operating environment. Test harnesses should be easily reconfigurable to accommodate changes in the list of system inputs and outputs.

7.8.3 Tools for creating, editing, and maintaining documentation,

7.8.4 Tools to keep track of time and resources invested in each version of the prototype, and in each component of the system,

7.8.5 Graphics tools for drawing flowcharts, data flow diagrams, state transition diagrams, etc., and

7.8.6 Computer-aided design programs for designing hardware, and producing engineering drawings.

## 8. The RP Loop

8.1 In RP, developers and users iterate through a series of *analysis—synthesis—evaluation* cycles in which the prototype becomes more and more like the required system. The basic RP process is shown in Fig. 3. One needs to use the RP loop to learn it. One can learn *about* it from a guide like this, but one must actually *do* it to learn it. The looping process starts from the initial definition of, and uses tools selected for, the project and the system. Analysis, synthesis, and evaluation meld together in the looping process; while each cycle through the loop is distinct, activities inside the loop are combined and integrated. Functional requirements and system design evolve as the prototype evolves. Design documentation can be minimized because the prototype itself is the best representation of the current state of requirements and design. (However, enough documentation must be maintained to understand the system.) User documentation evolves as part of the prototype. When the prototype is close enough to the required system, iteration stops and the project moves on to the implementation design and system assembly phases (Section 9). A prototype may evolve into an operational system, it may be used for experiments to learn about the behavior of some aspect of an operational system, or it may be used as an exact behavioral specification for an operational system.

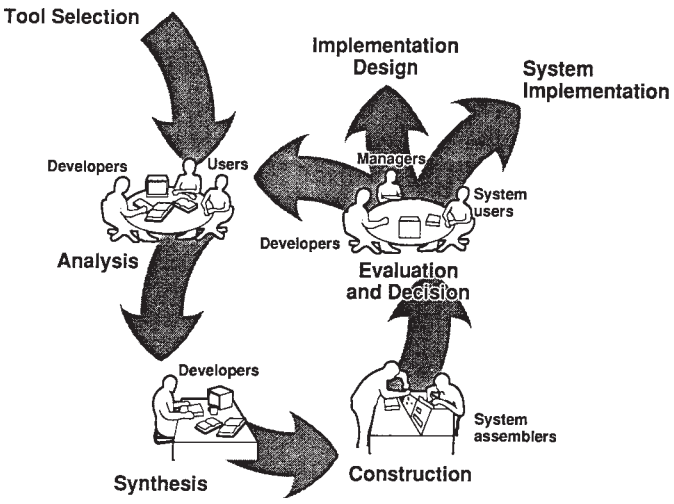


FIG. 3 The Rapid Prototyping Loop

8.2 *Analysis*—The initial analysis leads to selection of appropriate tools for learning more about the problem and for developing the required system. In each cycle through the RP loop, the analysis becomes more and more detailed about some aspect of users' needs and required system behavior. In each cycle, increasing demands are placed on a larger number of quality factors. What is most important is that users and developers communicate freely and often, so that they share a common growing understanding of the requirements and the system. Analysis in each cycle should be sufficient to synthesize the next version of the prototype. The analysis is never exhaustive and it always leaves some questions to be answered in the next cycle. The RP process ends when the questions remaining either already have known answers, or do not need answers for the system to be successful.

8.3 *Synthesis*—The first version of the prototype is guided by the initial analysis and the tools selected. A system developed by RP should start with an open architecture, that is, a structure and design style in which things can easily be added, changed, and removed. Avoid early integration and optimization; it is difficult to change later. Emphasize flexibility at the beginning. Structuring the system into subsystems is part of the iterative process.

8.3.1 Consider a number of options in each cycle and select one to be refined in preference to the others. Keep track of the choices and the earlier versions so that they can be recovered when it becomes necessary to backtrack.

8.3.2 In each RP cycle, analysis leads to synthesis of a stepwise modification of the open system architecture which exhibits behavior that can clearly be related to specific requirements.

8.3.3 Restructuring is sometimes necessary. If the RP process leads to deterioration of the logical modular structure of the system, affected parts should be redesigned to correct the problem. When components become obsolete because of evolution or because of design changes or errors, they should be thrown away; they should not be repeatedly patched or mended to make do. When the effort to restructure the system

can be recovered later because the new design works better than the old one, restructure to correct design problems.

8.4 *Construction*—Build the newly synthesized version. New components, concepts, designs, features, or structures are created quickly, tested immediately, fit into the rest of system, and tested with working pieces from earlier versions. The emphasis here is on speed.

8.4.1 Complete versions of the prototype should be delivered in each cycle. Additions and modifications to earlier versions are combined with unchanged parts to produce a new version.

8.4.2 Assuming (*1*) that the prototype is not pushing the state of the art in performance, and ( *2*) that the tools and approach that are selected are capable of providing adequate performance, leave optimization for later. When the prototype is finished, components that dominate its performance can be identified and optimized. These usually amount to a small percentage of the total components of the system. If performance becomes a large enough concern that optimization of the prototype is necessary, then re-evaluate the approach, because the prototype is not likely to evolve into a successful system. Go back and either redefine the project or start again with different tools or a different approach.

8.5 *Documentation* serves some of the same functions in a RP project that it serves in conventional system development (see Guide E 627), but there is less of it and the types are limited. Documentation communicates with people who are separated from the original development by time or distance. It tells people what they need to know (*1*) to continue developing the system, (*2*) to use the system, and ( *3*) to maintain the system. Some documentation is an integral part of the evolving prototype. Documentation is a tool which permits other people to work with the system when the original developers and users are finished.

8.5.1 Documentation should be prepared in each cycle through the RP loop. It should be done within the loop because if it is left to be done later, the probability of its being done at all is small.

8.5.2 *Documentation Requirements*:

8.5.2.1 *User's Manual*—If the system is to be used with a user's manual, that document should evolve as part of the prototype.

8.5.2.2 *On-line Helps and Messages*—If the user-system interface is being prototyped, messages to guide the user and on-line helps should be prepared and updated in each cycle.

8.5.2.3 *System Development Notes*, about decisions taken, options considered and set aside, and things that may have to be changed, should be prepared and reviewed (see 8.6.3). Developers should keep notes in a machine-readable journal or notebook.

8.5.2.4 *Engineering Drawings*, of hardware components should be prepared (*1*) when hardware design is not apparent by visual inspection, or ( *2*) when components are to be built outside the project team. A computer-aided design system should be used to create and uptake engineering drawings.

8.5.2.5 The best documentation to describe software module designs is in the form of embedded comments at the software design level where maintenance will be performed. This is usually the source code but it may be at a higher level if CASE tools are used to generate and change the source code (see 7.3.3). Development notes (8.5.2.3), graphical representations, and other documentation may also be needed. Code and documentation should be reviewed during each cycle (see 8.6.3).

8.5.2.6 As the project proceeds, the prototype itself is the most important representation of the system design, not a separate design document. However, enough design documentation should be maintained to understand the system.

8.5.3 Whenever possible, use graphics that communicate more efficiently than words.

8.5.4 The tools selected for the project should include good documentation tools. Use software for creating, editing, maintaining, and distributing documents. Programs that extract or generate documentation from high-level design languages or source code are especially useful.

8.5.5 Update the documentation during each cycle. In deciding when to uptake documents, divide them into two categories: documents related (*1*) to system use, and ( *2*) to system design. User documents should be updated before each evaluation. Design documents should be updated as the prototype changes.

8.6 *Evaluation*—People use the current version of the prototype to assess how well it meets their needs, to identify problems, and to propose additions or changes. Feedback to the developers should be immediate. Determine both what is right about this version, that is, what should be preserved, and what is wrong with this version, that is, what should be changed.

8.6.1 This part of the RP process is different from anything in conventional system development. This is the heart of successful RP, the fundamental strength of the method. Users interact with the prototype and react to concrete examples of its behavior. They learn what it can do, and they interact with developers to propose changes. Experience with actual system behavior helps users articulate statements of their needs. Users articulate likes and dislikes, and developers modify the prototype to accommodate them.

8.6.2 RP is fundamentally a learning process. From observing and experiencing reactions to the prototype, developers and users learn what is needed next. They recognize additional changes or extensions to make the prototype fit the users' needs better. In each cycle, users and developers learn more about users' needs, what needs are satisfied by the present version, how well they are satisfied, and how to accommodate additional or newly recognized needs.

8.6.3 Review new code, development notes and documentation during each cycle. An appropriate time to review is just before evaluation begins.

8.6.4 Formal reviews are helpful in some RP projects, and are one means of formalizing the process of making critical decisions. Evaluation reviews could cover:

8.6.4.1 Suitability of the current version for its intended use,

8.6.4.2 Quality of the current version,

8.6.4.3 Adequacy of the current documentation,

8.6.4.4 Potential improvements,

8.6.4.5 Potential generalizations,

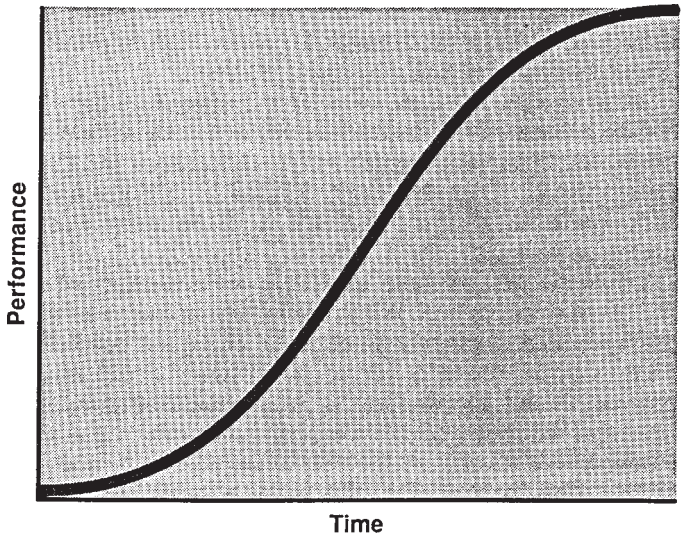


**FIG. 4 Typical Learning Curve**

8.6.4.6 Ways of restructuring the system to reduce complexity,

8.6.4.7 Directions for the next version, and

8.6.4.8 Whether to generate another version of the prototype or to move on to implementation.

8.7 There are a number of factors that affect user satisfaction with a prototype.

8.7.1 If the users who work with a prototype represent a larger population of potential users, then it is necessary to compensate for their learning curve by adding new people to the pool of users as the prototype matures.

8.7.1.1 A first-time user of a prototype is at the bottom of the S-shaped learning curve (Fig. 4), where great effort produces small improvements in performance. A person who is struggling to learn may ask for a large number of changes. A user who sees additional versions of the prototype builds on earlier experience and learns more about it. A user may stop asking for changes when the prototype seems just good enough, because additional changes would require additional learning effort. The prototype can thus become tailored to one or a few users' preference, but not necessarily to the best performance of the target population.

8.7.1.2 If the developers work with a series of new users, then they develop the prototype in a way that is appropriate for people at the bottom of the learning curve. If the prototype is never used by people who have experience with it, then the developers never see how experienced people interact with it.

8.7.1.3 If the group of users contains people who are at different places on the learning curve, then the developers get a more complete picture of how the learning curve affects user performance with the prototype.

8.7.2 The performance or functions of a first version prototype may lead inexperienced users to unrealistic conclusions about the eventual results. The limitations and goals of each prototype version should be discussed with people before they use it. Expectations need to be managed.

8.7.3 The Hawthorne effect[7] is a phenomenon in which any change in experimental variables appears to improve the performance of human subjects, because the phenomenon that is producing the change is not being measured. In the case of RP, it can cause any change in the prototype to produce an apparent increase in user satisfaction, whether or not the change is really helpful. It can be overcome by promoting straight communication between users and developers, and by looking only at changes in user performance between versions of the prototype. Comparisons against performance before the prototype was introduced should not be considered. If developers suspect that Hawthorne effect is present, then they can check for it by presenting different versions of the prototype to different users in different orders.

8.7.4 The *glitter of new toys* is related to the Hawthorne effect, but it affects early versions of the prototype. Users may be so taken with the prototype that they don't want anything changed from the first version, even though the developers know that it needs to be improved. In order to avoid this effect, each early version can contain some obvious flaw that users cannot ignore. This encourages them to propose changes.

8.7.5 After the *glitter of new toys* wears off, a reverse Hawthorne effect may appear in which no changes in the prototype increase user satisfaction. This means that it is time to stop RP and to move on to the implementation design and system assembly phases (Section 9).

8.8 *Iterate Rapidly and Cheaply*—Optimize time and cost to produce each new version. The process of cycling through the RP loop is carried out iteratively, and the prototype should converge to the operational system. Requirements and designs are either validated or refuted quickly. They can be changed quickly to reflect a growing understanding of the system. The number of iterations depends on available time and funds as specified in the project definition. Each iteration should bring the prototype closer to the operational system.

## 9. Implementation Design and System Assembly

9.1 When RP is finished, it may be appropriate to develop an implementation design for the operational system using Guide E 622, Section 9 and to assemble, install, and test it. (See Guide E 622, Sections 10 and 11.) The prototype is sometimes replaced by the operational system; sometimes the prototype becomes the operational system.

9.2 *When to Keep and When to Replace*:

9.2.1 One of the dangers of the RP approach to system development is that an inadequate prototype can sometimes be forced into operational service when it should be rebuilt from scratch. This postpones the immediate cost of rebuilding in favor of large costs for future maintenance. Before starting a RP project, everyone involved with it should agree either that the prototype is to evolve into the operational system or that the operational version is to be rebuilt from scratch. If this cannot be decided when the project starts, then the decision criteria should be stated in the project definition agreement (see 6.4). It

[7] Roethlisberger, F. J., and Dickson, W. J., *Management and the Worker*, Harvard University Press, Cambridge, MA, 1939.

may be appropriate to rebuild only parts of the prototype instead of the whole system.

9.2.2 Even when the original intent of the project was to evolve the prototype into an operational system, it may be appropriate to revise the project definition at some point and to relabel the prototype as experimental or build-and-replace. This decision should be taken if it is learned that the system architecture or RP tools are not right and cannot be made right without starting again. It is better to change the RP approach and the project definition to incorporate what is learned, than to keep plowing ahead toward likely failure of the operational system.

9.3 *When the prototype becomes the operational system*, there is no implementation design phase because the prototype itself does not need to be redesigned. The project moves directly into the system assembly, installation and testing phase, for optimization and refinement of the prototype.

9.3.1 Aspects of the system that are neglected for the sake of rapid cycling through a series of versions of the prototype need to be brought to an acceptable level. Optimization should have been left for this phase, so the system should be optimized to run efficiently. Backups and controls may be needed to make the system acceptable for operational use.

9.3.2 If the system users include people who were not part of the RP team, then they need to be trained to use the new system. Guide E 625 covers training for system users, end users, supervisors and managers.

9.3.3 If the prototype is to replace an existing system, then it is necessary to cut over from the existing system to the new one. This may involve retraining users, converting data files, revising manual and backup procedures, etc. Even though the new system was developed by RP, cutover should be accomplished with the same detailed planning as in a conventional system development project.

9.3.4 A prototype of the user and system documentation should evolve along with the prototype of the system itself. At the end of each *analysis—synthesis—evaluation* loop, the documentation should be updated to the current level of the prototype.

9.3.5 A RP project does not generate the same sort of documentation as a conventional project. The nature of the system, the complexity of the development effort, the relationship of users to developers, and the need for system maintenance all influence the quantity and content of project documentation. Additional effort may be needed at the end of a RP project to produce documents that would be produced during the course of conventional system development.

9.3.6 Documentation Guides E 627 and E 1029 and Specification E 919 and ANSI/IEEE 1063 apply to the operational system, even if it is developed by RP. If a system developed by RP is to be maintained, then the maintenance documentation described in Specification E 919 is needed. If the system is to be used by anyone other than the users with whom it was originally developed, then the two levels of user documentation described in Specification E 919 are needed. At the end of the project, go back and finish the needed documentation that was not completed as part of the prototype.

9.4 *Reworking a prototype* into a polished well-built system is easiest if the conceptual structure of the prototype is clear. A prototype of limited scope should be developed with a view to its future extension into a complete system. A prototype usually needs to be scaled up to fit the whole problem domain.

9.4.1 The prototype may be extended by adding secondary functions that were ignored for the sake of rapid cycling.

9.4.2 Inefficient components of the prototype should be changed or replaced to yield the required performance efficiency.

9.4.3 One RP strategy is to provide a user-system interface that is useable only by the development team. If this strategy has been taken and it is necessary to improve the interface, use RP to rework it (see 7.4).

9.5 If the system is to be rebuilt from scratch, the prototype serves as a specification for its behavior. In this case, the project moves into an implementation design phase for the operational system. One advantage of RP over conventional system development, is that a prototype can be a more precise specification than a written document.

## 10. Project and System Evaluation

10.1 After the system is operational, both the system and the project are evaluated using Guide E 622, Section 11. This evaluation collects the lessons of the project and makes them available for future use. In the system evaluation at the end of a conventional development project, the operational system is compared against its functional requirements to determine how well the requirements are met and how much capacity exists for growth. Because functional requirements are implicit in the prototype, system evaluation compares what is delivered against the perceived (but undocumented) needs of its users, at least some of whom participated in its development.

10.2 A RP project is a learning activity for everyone involved with it. Project goals may include learning the users' real requirements, learning whether a new idea or design is feasible, learning the capabilities of new tools or computers or systems, learning how to build some component or feature of a system, or learning how to use RP effectively to develop computerized systems. The project evaluation needs to crystallize and collect the lessons learned, in a form that is useful to people other than those who participated in the project.

10.3 The tools selected for a RP project have a profound effect on its outcome. The project evaluation should include an evaluation of the tools used, and recommendations about their future improvement and use.

10.4 If a prototype is used as a specification for building the operational system, then the system evaluation procedure of Guide E 622, Section 11 can be followed with only slight modification. The prototype itself provides the functional requirements against which the operational system is evaluated.

10.5 If a prototype evolves into the operational system, then the users' requirements should be determined from interviews and questionnaires. Evaluators are likely to hear most from users about aspects of the system that are less than fully satisfactory. In one respect, this makes evaluation easy because the users' comments focus on things that the evaluators should find. In another respect, however, it makes evaluation difficult

because the evaluators' attention is directed away from things that work well, which are usually more important and valuable than the problems.

10.6 The evaluation report should document the functional requirements against which the system is evaluated, and the same requirements should be reused for retrospective and final evaluation (see Guides E 622, Section 11 and E731). The prototype should also be saved for reuse in later evaluations. It should not be thrown away.

## 11. Rapid Prototyping in Large Projects

11.1 RP can be used to explore system feasibility and overall system architecture, to develop subsystems or components that are not well understood, and to test different designs of system elements that are critical to success. In a large project, several RP subprojects may be active at the same time, with each one proceeding around its own independent RP loop. Subprojects do not need to be synchronized until the end of the RP activity, when the implementation design or system assembly phase starts.

## 12. Keywords

12.1 computerization; computerized system; documentation; evaluation; functional design; functional requirement; implementation design; project definition; rapid prototyping; subprojects; system design; system development

# APPENDIXES

**(Nonmandatory Information)**

## X1. CONVENTIONAL SYSTEM DEVELOPMENT METHODS

X1.1 The conventional process for developing a computerized system is shown in Fig. 1. It includes the following six phases (see Guide E 622 and Terminology E 1013):

X1.1.1 Project Definition (see Guide E 622, Section 6),

X1.1.2 Functional Requirements (see Guide E 622, Section 7),

X1.1.3 Functional Design (see Guide E 622, Section 8),

X1.1.4 Implementation Design (see Guide E 622, Section 9),

X1.1.5 System Assembly, Installation, and Testing (see Guides E 622, Section 10 and E625 and Practice E 731), and

X1.1.6 System Evaluation (see Guide E 622, Section 11).

Documentation is an important product of each of the six phases (see Guide E 627). It is prepared as an integral part of the project, not as a separate activity. Documentation carries information about the requirements to people who design the system, and additional documentation carries information about the design to the system builders. Requirements analysis, design, and implementation are separate activities, which produce documents that are used in other activities.

X1.2 Conventional methods have been proven to increase the probability of success in developing computerized systems, but they have some problems.

X1.2.1 Conventional system development methods expect that most activities will be done right the first time they are undertaken. While the conventional method described in Guide E 622 allows for backtracking and cycling through the six phases, the cost of correcting errors and making changes increases as the project moves farther from defining requirements and closer to the operational system. It is much more expensive to correct an error, or to make an enhancement in an operational system than in a system that is still being designed.

X1.2.2 In conventional system development methods, the functional requirements, functional design, and implementation design are produced as written documents which are passed to the users for comments, changes, and approvals. These documents provide many opportunities for error or misunderstanding. Functional requirements are the most critical and problem-prone area of system development.

X1.2.3 For first-of-a-kind systems, what the system must do and how it can be produced are not usually understood when the project starts. Some system development projects are entirely straightforward, but many projects have elements that are only partially understood. In custom software development, systems are usually delivered late and they often prove not to be what the customer wanted.

X1.2.4 The time period between the early stages of requirements analysis and delivery of an operational system is sometimes long. Users' needs may change significantly during the project; sometimes the changes are so significant that the delivered system is of no use at all.

X1.3 The conventional system development method described in Guide E 622 shares several important characteristics with the rapid prototyping method described in this guide.

X1.3.1 Both methods are centered on the needs of the system users.

X1.3.2 Both methods involve the determination of functional requirements but RP iteratively refines the requirements.

X1.3.3 Both methods involve specifications. RP refines the specifications in a working model and the conventional method refines them in written form.

X1.3.4 Both methods involve testing, but RP uses immediate testing and correction.

X1.3.5 Both methods proceed through several phases. The functional requirements and functional design phases of the conventional method are replaced by the RP loop.

X1.3.6 Both methods emphasize documentation. In RP, the prototype itself is the most important representation of the design, not a separate design document.

X1.3.7 Both methods can proceed in cycles. The conventional method allows for backtracking and repetition of earlier phases, while RP is an inherently cyclic process.

X1.3.8 RP may initially sacrifice efficiency and robustness of the prototype, but it can provide these qualities at a later stage in the project.

## X2. DECIDING TO USE RAPID PROTOTYPING

### INTRODUCTION

RP is most effective for solving ill-structured system development problems. A well-structured RP approach to a difficult or ill-structured problem ensures that the maximum benefit will be achieved and that, if appropriate, the prototype can evolve into an operational system.

X2.1 *Advantages of RP:*

X2.1.1 It is natural to build a model and try it. Inexperienced system developers usually fall into some form of prototyping if they are not taught a conventional system development method.

X2.1.2 It provides a structure for users and developers to work together to define requirements and to design, implement, and test each version of a developing system.

X2.1.3 It provides a framework in which to build the system.

X2.1.4 It is not necessary to do things right the first time.

X2.1.5 It quickly produces a working model which, in turn, can produce better understanding of the nature of the system, better estimates of time and effort to produce an operational system, quicker validation of design concepts, and faster testing and revision of the design.

X2.1.6 It leads to better understanding of the functional requirements because:

X2.1.6.1 Users may not know what they need or want when the project starts.

X2.1.6.2 Developers may not understand the users' requirements.

X2.1.6.3 If the initial version does not reflect the users' true needs, the needs can be accommodated in later versions.

X2.1.6.4 It is not necessary to have full agreement about all system capabilities at any early stage in the project.

X2.1.6.5 Each version encourages users to think carefully about needed and desirable characteristics.

X2.1.6.6 The prototype allows users to react to actual system behavior.

X2.1.6.7 The end product is a more tangible, stable reflection of requirements.

X2.1.7 It can produce a better user-system interface.

X2.1.8 A prototype can be easily modified.

X2.1.9 It does not require accurate time estimates at the start of the project.

X2.1.10 It can reduce system cost and produce a better system from a better defined target.

X2.1.11 The rapid feedback can help inexperienced people (both developers and users) to learn quickly.

X2.1.12 It uses experienced people effectively.

X2.1.13 It is likely to shorten overall system development time.

X2.1.14 It is likely to produce an operational system which is modifiable and testable.

X2.2 *Disadvantages of RP:*

X2.2.1 It can produce an inefficient system. A prototype may not reveal how a system will behave when pushed to extremes of performance (for example, heavily loaded, buffers exhausted, displays saturated with data) or how it will interact with other system components.

X2.2.2 It can produce a system that lacks portability or generality. The prototype system may not be useable outside the hardware, software, or organizational environment where it is produced.

X2.2.3 It can produce a system that is not complete.

X2.2.4 It can give users a wrong perception of the operational system. A prototype does not give users a true picture of what it is like to live with a system for the long term.

X2.2.5 An RP project may use inappropriate tools or select an incorrect starting point.

X2.2.6 An RP project may be slow to find an optimum design.

X2.2.7 RP can make it difficult to maintain a tight development schedule.

X2.2.8 Tools needed for RP may be costly.

X2.2.9 A prototype can be forced into serve as an inadequate operational system.

X2.3 *RP does not automatically solve these problems*, which are pitfalls in both conventional system development and RP, and which may be aggravated by RP.

X2.3.1 The resulting system may be difficult to maintain.

X2.3.2 The resulting system may not be adequately documented.

X2.3.3 The resulting system may lack robustness and reliability.

X2.3.4 The resulting system may lack an adequate user-system interface. The interface may be useable only by the development team or it may lack full error handling and recovery.

X2.4 *RP should be used under the following conditions:*

X2.4.1 A good deal of uncertainty exists about the requirements or how to build the system.

X2.4.2 Both the risk and the cost of project failure (direct monetary costs or time and effort) are low to moderate. However, RP can be used to explore feasibility of different designs in high-risk projects.

X2.4.3 The system has a heavy requirement for software.

X2.4.4 The prototype is small enough to be developed by a team of no more than seven users and developers.

X2.4.5 There is a close working relationship between users and developers.

X2.4.6 Each RP phase of the project can be completed in no more than nine months.

X2.5 *Avoid RP Under the Following Conditions:*

X2.5.1 The project is large. However, RP can be used to develop the overall architecture and subsystems and components of large systems.

X2.5.2 A large data base must be created in the project before the prototype can be run.

X2.5.3 The risk or cost of failure is high (see X2.4.2).

X2.5.4 Errors can threaten the safety of life or property. For example, it would not be appropriate to use RP to develop the process interface for a real-time control system in a hazardous environment. (It might be appropriate to use RP to develop the user-system interface for the process, if the prototype was not connected to the process interface.)

X2.5.5 The system is well-understood, well-defined and well-specified. Just go ahead and build it in one version.

## X3. WHEN PROTOTYPING IS LESS THAN RAPID

X3.1 *Rapid* in RP means that the time between successive versions of the prototype is short. It should be short enough that (*1*) both users and developers can remember how each version relates to the previous one, (*2*) user requirements do not change significantly while a version is being developed, (*3* ) user interest is maintained, (*4* ) the prototyping team will remain in the project through the RP phase, and (*5*) total time to develop the system is acceptable. (Expected project duration should be stated in the project definition. See Section 5 and Guide E 622, Section 6.) A few days between versions is adequate and a few weeks may be acceptable.

X3.2 If the time needed to produce a new version is long, then steps should be taken to supplement the memories of development team members. Arrange for both the previous and current versions of the prototype to be simultaneously available for evaluation. This makes it easier to compare features and behaviors that have changed.

X3.3 If two versions of the prototype cannot be simultaneously available for evaluation, then it is necessary to document what needs to be changed in the next version, how it works in the current version, and how it is to be changed. Documentation is less reliable than direct comparison because features that are not supposed to change may, in fact, change, with no documentation to describe the previous behavior.

X3.4 Disagreements among members of the development team about how things worked in earlier versions or how they were to be changed usually indicate (*1*) that new versions are not being developed rapidly enough, (*2*) that simultaneous versions should be provided, or (*3*) that more documentation is needed for each prototype version.

X3.5 If prototyping does not work well, abandon it and produce the system using a conventional system development method (for example, Guide E 622) with full documentation of requirements and design. It is better to change the approach and the project definition than to keep plowing ahead toward likely failure of the project.

## X4. RELATED LITERATURE

*ACM Software Engineering Notes*, (Special Issue on Rapid Prototyping), Vol 7, No. 5, December 1982.

Agresti, W. W., editor, "New Paradigms for Software Development,'' IEEE Computer Society Press, Washington, DC, 1986.

Fisher, G. E., "Application Software Prototyping and Fourth Generation Languages,'' NBS Special Publication 500-148, 1987. Available from the Superintendent of Documents, U.S. Government Printing Office, Washington, DC 20402.

IEEE 1074, Software Life Cycle Processes, Institute of Electrical and Electronics Engineers, Inc., 345 E. 47th Street, New York, NY 10017.

Naumann, J. D. and Jenkins, A. M., "Prototyping; The New Paradigm for Systems Development,'' *Management Information Systems Quarterly*, September 1982, Vol 16, No. 3, pp. 29–44.