Publication Funded by
Bureau of BJA Justice Assistance
U.S. Department of Justice

Structured Systems Development Guidelines

125624

SEARCH Group, Inc.
The National Consortium for Justice Information and Statistics

SGI

125624

Structured Systems Development Guidelines

June 1990

125624

SEARCH Group, Inc.
The National Consortium for Justice Information and Statistics
7311 Greenhaven Drive, Suite 145 • Sacramento, CA 95831
(916) 392-2550 • FAX (916) 392-8440

U.S. Department of Justice
Office of Justice Programs
Bureau of Justice Assistance

This document was prepared by SEARCH Group, Inc. under Cooperative Agreement No. 87-SA-CX-K086, provided by the Bureau of Justice Assistance, U.S. Department of Justice. The points of view or opinions stated in this document are those of SEARCH Group, Inc. and do not necessarily represent the official position or policies of the U.S. Department of Justice.

Bureau of Justice Assistance
633 Indiana Avenue, N.W., Washington D.C. 20531
(202) 272-6838

Acknowledgments

Structured Systems Development Guidelines was completed under the auspices of SEARCH Group, Inc., the National Consortium for Justice Information and Statistics. Gary L. Bush is the Chairman and Gary R. Cooper is the Executive Director of this nonprofit consortium of the states which is dedicated to improving the criminal justice system through information technology.

Work on *Structured Systems Development Guidelines* was supervised by David J. Roberts, SEARCH Deputy Director, Programs. *Structured Systems Development Guidelines* was written by Thomas Wilson, Senior Writer, and Noel Morgan, Manager, Technical Services, formerly of SEARCH Group, Inc. Editorial and formatting assistance was provided by Judith A. Ryder, Director, Corporate Communications and Jane L. Bassett, Publishing Assistant. The project was conducted under the direction of R. John Gregrich, Chief, Corrections Branch, Bureau of Justice Assistance, U.S. Department of Justice.

TABLE OF CONTENTS

Introduction

Successful software development projects invariably share a common element — they are carefully planned and controlled by structured guidelines. The development of software is an extremely complex process, one that must integrate technical, managerial and administrative functions into a coordinated, systematic approach.

Structured Systems Development Guidelines provides an overview of the process for developing computer software, including practical management and technical guidelines for planning, managing and developing software. The *Guidelines* define the work to be done, a methodology for accomplishing it, the project deliverables, and the documents to be produced. It is directed at both administrators and managers who are responsible for oversight and control of the development effort, as well as technical personnel responsible for the actual development of the software program.

The text of the *Guidelines* is organized by the four phases of the software development process: planning; design; coding and testing; and operation and maintenance. Within each phase, there is a narrative overview of the development process, followed by detailed formats for the documentation that must be produced in each phase. The narrative overviews are designed to provide managers with an understanding of the development process, as well as practical advice on critical elements necessary to ensure the success of the project. The document formats are included in the *Guidelines* to suggest to managerial and technical personnel sample layouts and the typical content of key documents.

The *Guidelines* are designed to impose order on the project and to make all activities in the development process visible to administrative, managerial and technical project personnel. It is crucial that all phases of the project be clearly defined and documented. Each phase can then be assessed for quality and validated through testing and formal reviews to ensure compliance with development objectives. Accordingly, *Structured Systems Development Guidelines* stresses three elements that must be present in software development if the project is to be successful:

1. The process must be a *disciplined* one, evolving through systematic phases. Technical guidelines and methods must be strictly followed through software project planning, requirements specifications, designing, coding, testing, and operation and maintenance of the system.

2. The process must be *visible* to both management and technical personnel at all times to ensure that all phases of the process are executed according to a disciplined method and that the requirements specified by the end user are completely met. Therefore, a set of formal documents delineating the entire process and methodology — from planning through implementation — must be created.

3. The process must provide *quality control.* Each phase of the process must be guided by a methodology that tracks, reviews, accommodates changes and validates the development of the system. The process is not disciplined, visible and quality controlled if it is not in writing. Documentation is essential. Without it, the process is invisible and, almost surely, there will be errors, cost overruns, and client dissatisfaction. The project, therefore, must be articulated in a project plan that incorporates all methods, procedures, schedules, milestones, reviews, costs and resources. Each phase of the project must be documented comprehensively. Most important, the design of the software must be complete, down to the level of the smallest subroutine of its programs. Structuring and documenting the process and the design, in essence, allows the project to be managed. Management is the key to control and, ultimately, to the successful operation of the system and to user satisfaction.

The techniques contained in *Structured Systems Development Guidelines* will enable project personnel to:

- Understand and control the development process;
- Define, document and implement a project plan;
- Establish organizational roles and responsibilities;
- Establish formal lines of communication among the administrators, managers, technicians and the end users;
- Identify and allocate technical resources;
- Develop realistic and accurate costs and schedules and control them throughout the development process; and
- Design, validate and deliver a quality system that is responsive to the end user's needs and requirements.

Project Phases

Software development has four stages: planning, design, coding and testing, and system operation and maintenance.

The software development process has four sequential stages: planning, design, coding and testing, and system operation and maintenance. Changes are an inevitable part of the software development process. Despite the most rigorous planning efforts, refinements and better development techniques may nevertheless be identified once the actual development is in progress. A change in one small part of the system, however, may significantly affect other parts of the system. Accordingly, the development cannot be a single linear process; it must also be iterative, repeating and refining steps to ensure the integrity of the system.

Planning

In the Planning Phase, the development team defines *what* the end user wants and *how* the development team will accomplish it. The functional requirements of the end user are identified and translated into a computer software environment. The goals of the planning stage are to ensure that the end user's requirements for the proposed system are clearly understood by the development team, and that the system's capabilities and limitations are clearly understood by both the developers and the end user.

The purpose of the Planning Phase is to identify the user's needs and to create a blueprint for a computer solution. This phase is executed in five parts.

1. The **Requirements Analysis** identifies the needs of the user, which are then defined and documented in the End User Requirements.
2. The **System Specifications** provide a global design of the proposed computer software solution, and include necessary information on the hardware upon which the software will run.
3. The **Preliminary Project Plan** is an initial assessment of the nature, scope, resources and costs necessary to complete the project and is intended to give management the information it needs to determine if the project is feasible and to make the commitment to go forward.
4. The **Project Plan** is the formal strategy for accomplishing all work necessary to develop the system, including personnel resources, technical resources, schedules, budgets, managerial and administrative responsibilities and all other policies, procedures or techniques designed to keep the project on track and under control.
5. The establishment of **Change Control** provides a formal mechanism and protocol for requesting changes in the design of the proposed system, which not only allows project management and the technical development team to keep all development activity visible, but also ensures the integrity of the system design.

Design

In the Design Phase, the general concept of what the system must do, as specified in the Planning Phase, is formulated into a specific computer solution. The design team breaks the system down into discrete modules and groups of modules and determines how each module will perform its task.

The Design Phase has four parts.

1. The **Preliminary Design** is the "first cut" at the design of the system. As such, it provides the architecture, or model, of the software design as well as a description of the behavioral characteristics of each function of the software—that is, how the functional requirements will be realized in a computer environment.
2. The **Preliminary Design Walkthrough** validates the soundness of the Preliminary Design by having a review team of technical experts analyze it.

3. If the Preliminary Design is approved, the **Detailed Design** moves the process of the software design to the level of specifying the algorithms for each program and subroutine of the proposed software.
4. The final part of the Design Phase, the **Detailed Design Walkthrough,** validates the Detailed Design by having technical experts analyze the design.

Coding and Testing

In the Coding Phase, the programmers translate the software design into executable code for each module, groups of modules, and finally into a fully integrated system. During the coding process, continuous testing takes place in an iterative manner until each module, group of modules and the integrated system functions according to specifications. Once the system is integrated, its functionality and conformance to the user's requirements must be officially validated.

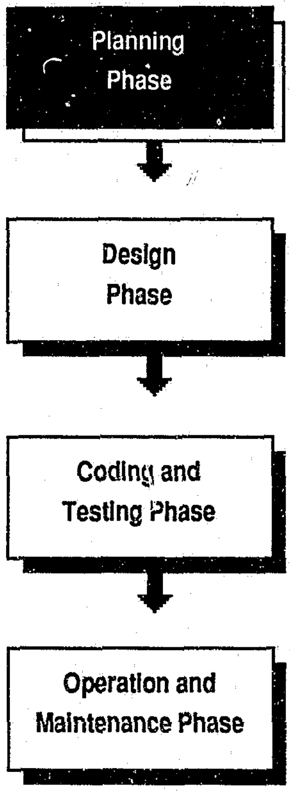
Coding and testing have six major parts. 1) Individual modules are coded in parallel by several programmers using standard development techniques. 2) Each module is then validated during a Coding Walkthrough. 3) Unit Testing follows in which each module is tested individually. 4) Integration Testing tests the modules as functional groups. 5) Validation Testing confirms that all modules assembled into a unified system perform according to design specifications and comply with all user requirements. 6) In Beta Testing, the system is installed at a test site to allow the development team to identify errors in an operational environment.

Operation and Maintenance

The Operation and Maintenance Phase installs the system in the end user's environment and provides support for the system, including correcting problems found only in operational use, enhancing the system at the request of the end user, and modifying the system as new versions of the software become available.

In Summary

The process and techniques presented in *Structured Systems Development Guidelines* are intended to establish a formal methodology for systems development. It is the methodology employed by the management staff and development teams at SEARCH Group, Inc., the National Consortium for Justice Information and Statistics, in the development of public domain criminal justice software systems. Regardless of the size and duration of the effort, it is most important that the process be implemented. Finally, there may be instances where parts of the process as delineated in the *Guidelines* do not fit a specific development project. In such cases, deviation from the process should occur only after an alternative has been deemed necessary and reasonable and after careful consideration of the possible effects. By utilizing the *Structured Systems Development Guidelines* methodology, an agency can develop realistic cost and schedule estimates, limit design flaws and programming errors and establish realistic testing procedures — all of which result in effective, high quality systems produced in a controlled and efficient manner.



Planning Phase

Requirements Analysis

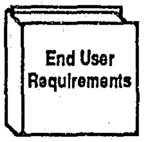
A Note Before You Start: *A software development effort begins after a decision has been made to develop a new system for a known end user or for a market where the need and applicability of the software is known. At this point, the end user usually has only a general concept of a desired system. The end user is responsible for performing (or having systems consultants perform) a needs assessment and systems analysis. These activities must precede the systems development effort because they verify the actual need for a software solution and identify in a preliminary manner the purpose and function of the development effort. Taken together, the needs assessment and systems analysis provide a preliminary concept of the intended system. In the actual development process, the development team will work with the end user to refine the system concept.*

The development process begins with a clear definition of the end user's requirements. To accomplish this, the development team must first see the problem from the user's point of view. This task is called the Requirements Analysis. During the Requirements Analysis, the development team works with the end user to identify the purpose of the system, problems in the existing manual or automated system that the end user is trying to overcome, and improvements the end user would like to make. In essence, the Requirements Analysis attempts to understand exactly what the end user really needs. A proposed system solution can be responsive only if the user's problems and needs are clearly and completely understood.

In the Requirements Analysis, therefore, the development team must be able to identify all of the functions that the user *wants* the proposed system to perform. In conducting the Requirements Analysis, the development team is attempting to define only the end user's *functional* requirements. What the user wants — that is, the functional requirements of the system — are delineated in a document called the Requirements Specifications. This document represents a mutual understanding and formal agreement between the user and the developer of the nature of the problem and the functions that need to be performed by a software solution.

With a knowledge of what the user wants, the development team generally determines *how* it could be accomplished as a computer solution. This translation of what is needed into a computer solution is delineated in the System Specifications document. At this stage, the development team need only be concerned with a global or conceptual overview of the system solution, rather than defining the actual architecture of the software, since that is the task of the Design Phase. Taken together, the Requirements Specifications and System Specifications form the conceptual understanding of the proposed system. They are, in essence, the blueprint for the design of the system.

The Requirements Specifications produced in this phase must be reviewed by both the end user and the development team to ensure all the requirements have been identified and that both parties are in complete agreement about the functions that are desired in the proposed system, as well as the feasibility of the development request. The development team needs to maintain close contact with the end user during this phase of the process. Time and care spent in this phase will ensure satisfaction with the final product.



Requirements Specifications Document Format

1.0 Overview

Provide an overview of the nature and scope of the end user's problem, including the user's perceived needs and expectations for the development project. Describe the current systems environment, whether it be manual or automated. Include historical information, such as the nature and kinds of work the end user performs, the tasks and deliverables that are associated with the existing systems, prior computer projects that may influence the current systems development effort, and the current level of computer skills possessed by the end user's staff.

If there is an existing system — manual or automated — fully describe its functions. A clear understanding of the functions, strengths, and weaknesses of the existing system is critical to the design of a replacement system. Specify the maximum limits for the time and budget of the development effort.

The Requirements Analysis identifies the end user's functional requirements.

2.0 Requirements of the Proposed System

Identify the functions the existing system does not satisfy, and describe how the proposed system will address those deficiencies.

2.1 Performance Requirements

Specify the performance requirements of the proposed system. Performance requirements should be stated as specifically as possible, but in nontechnical terms, such as "to sort and create 1000 mailing labels per hour," or "to store 50,000 names in a database and to retrieve any given name from the database in less than 3 seconds." If properly identified and documented, the performance requirements become the baseline for testing and evaluation during the coding and testing phase.

2.2 Operational Requirements

Discuss the effect the operation, maintenance or support requirements may have upon the design of the proposed system. Include a discussion of training, hardware and software maintenance and technical support.

2.3 Compatibility

Discuss compatibility requirements between the proposed and existing system, and any limitations that will be imposed on the new system as the result of incompatibility. If applicable, discuss how the use of standard hardware and software may affect or minimize compatibility problems.

2.4 External Interfaces

Discuss interface requirements between the planned system and other systems. For example, there may be a requirement to format output so that it can be transmitted and used by another system. Define the interface required between the user and the proposed system. For example, the end user might require that system interaction use mouse-driven, color pull-down menus, or the layout of the user interface screens might be required to have the same formats as the existing system.

2.5 Internal Interfaces

Discuss all of the major functions required of the system and how each of these functions will interface.

2.6 Design Constraints

Discuss any constraints placed upon the design of the system because of performance or use requirements.

2.7 Existing Environment

Identify any existing equipment, programs or operating procedures that must be used in the new system. Identify the type, model and manufacturer of the computer system upon which the system must operate. Specify the number, size and type of peripheral devices required for the new system. Include items such as additional printers, PCs, terminals, hard disk drives, network adapters or other devices.

2.8 Growth Requirements

Specify the projected minimum and maximum yearly growth of the system for a five year period. Identify additional storage, processing, peripheral and additional user capacity which may be required during the five years. Describe how the proposed system will address these needs.

2.9 Audit Trail Requirements

Identify the end user's requirements for an audit trail of the system. Include user logs, transaction logs, and the operation of the audit features of the proposed system.

2.10 Documentation Requirements

Discuss the end user's requirements for system documentation. Include a User's Manual, an Administrator's Manual, Technical Specifications, Functional Specifications and a Test Plan. Taken together, these documents should enable the user to review the functions and operations of the system, install and maintain the system, understand in detail the technical and functional specifications of the system, and perform an operational test of the system to confirm that it functions as designed.

2.11 Documentation Prototypes

Identify the required type, content and format of any new forms or reports which are to be used as input or generated as output from the system. Examples of prototype forms will give the end user and designers a better understanding of the content of the forms.

2.12 New Bounds

Specify the required bounds to be placed on the new system. Include maximum capacities such as file sizes or number and type of records, maximum execution times or other convenient and verifiable limits.

System Specifications

The Systems Specifications define what the development team has contracted to produce.

Once the end user has defined the purpose and desired functions of the system, it is the task of the design team to translate this user definition into a system solution — that is, into hardware and software that will meet the user's needs. The user defines the need; the development team defines the solution in terms of computer capability. The translation from user need to computer solution involves limitations inherent in digital computers. That is, computers process information or perform functions within the parameters of digital processing. There are functions a computer can perform and functions it cannot. It is critical, therefore, that the development team work closely with the end user to achieve an understanding of the manner in which needs can be accommodated by a computer solution, including inherent limitations, and to come to complete agreement that such a solution is satisfactory. The System Specifications — the hardware and software specifications — define what the development team has contracted to produce.

At this point in the process, the purpose of the System Specifications is to specify the global system architecture. For example, if the Requirements Analysis has ascertained that the user needs a system to store, index, link and retrieve large and diverse quantities of information in the form of flexible report formats, then the development team has the initial information it needs to design the database management system software. Given the volume of information to reside in the system, the number of users and the frequency of inputs and outputs, the development team can create a data flow diagram of the information flow, specify the modules that will have to be developed, specify the interaction of the modules, and generally provide a conceptual design of the system. The distinction between this phase and the design phase is that the System Specifications defines the modules and their functions, whereas in the Design Phase the designers specify how each of the modules will work in a specific computer environment.

The System Specifications also describes the architecture upon which the software will run and the end user's current systems environment and operations, including the use, physical location, hardware and software interfaces, and other organizations or systems that will communicate with the proposed system. The hardware description is critical to the development of testing and performance verification procedures in later phases of the development process. As such, the preparation of this document should be detailed and complete.

If the proposed system is to replace an existing system, then an understanding of the flow of information through the proposed system is based in part upon a knowledge of the structure of the existing system. All existing computer-based systems, hardware subsystems, procedures and tools in existence must be carefully examined and evaluated to determine their influence on the development of the new system. The development team needs to identify where the new system will be located and any potential constraints which will be placed upon it due to its location or environment. There may be existing systems or subsystems which will call for specific interfaces or data organizations. There also may be related procedures and requirements, as well as other organizations that will use the new system, that will impose bounds or limits on the System Specifications.

After careful evaluation by the development team to ensure that all user required functions have been translated into a workable computer solution, the System Specifications document is approved and becomes the baseline for the software design.

System Specifications

System Specifications Document Format

1.0 Overview of System Specifications

Provide a brief overview of the System Specifications, describing the proposed hardware and software requirements.

2.0 Reference Documentation

List pertinent documents related to the system specifications. List all of the documentation and manuals for both the software and hardware. They will be used as references by the development team during the remainder of the project. Documentation includes operating system manuals, user manuals, technical manuals and reference materials, operation manuals and specification guides. In a Local Area Network (LAN) environment, include all of the manuals and references to the cabling scheme, network servers, workstation adapters, and the network operating system. In a UNIX-based environment, include the terminal reference manuals, operating system manuals, and manuals on all peripheral devices.

3.0 Software

Describe the software that will be required to interface with other software programs, utilities, or drivers, including:

- Operating system;
- Language(s) or database management development systems required to create the new system; and
- Application software with which the proposed system may need to be compatible (such as word processing or database software).

A diagram, such as the one in *Figure 1: Sample Interface Diagram* (below) should help identify relationships among the proposed software and any other software elements or applications in the same environment.

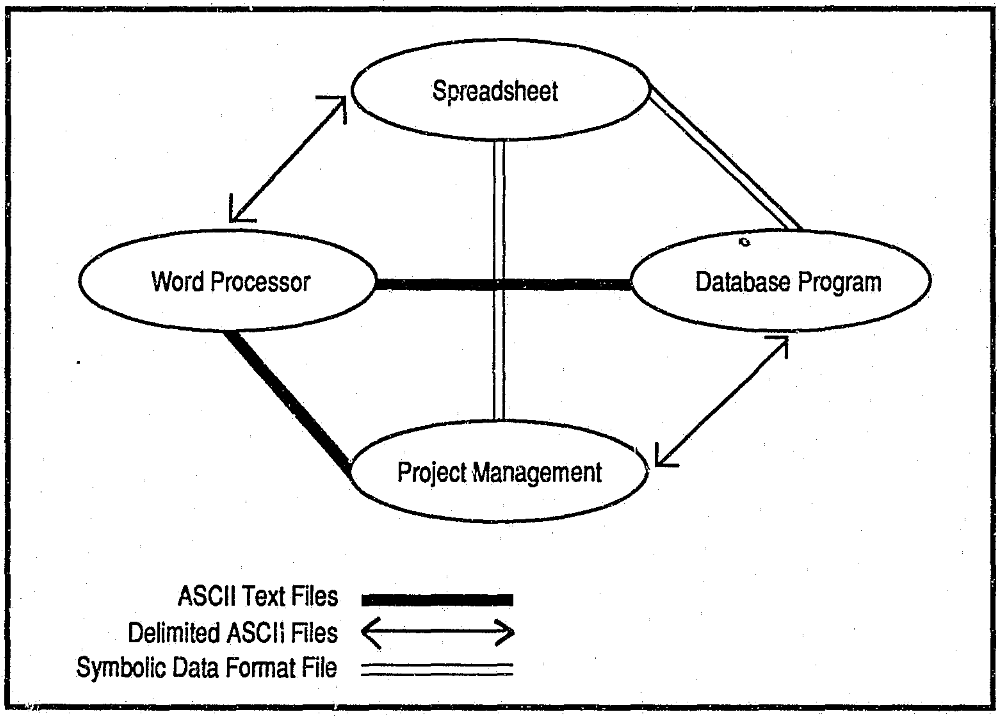


Figure 1: Sample Interface Diagram

3.1 User Interface

Describe the user interface of the proposed software. Include descriptions of items such as the use of color, sound, graphics and light pens, as well as on-line help, pull-down windows or computer aided training.

3.2 Overall System Diagram

Provide a diagram of the software modules and their relationship to each other and to external system elements.

3.3 Internal Data Flow

Provide a diagram showing the data flow throughout the system. It should depict the input and output of data, as well as sources of data which may be outside the programs to be designed. This type of chart is commonly known as a "bubble chart." (See *Figure 2: Sample Data Flow Diagram.*)

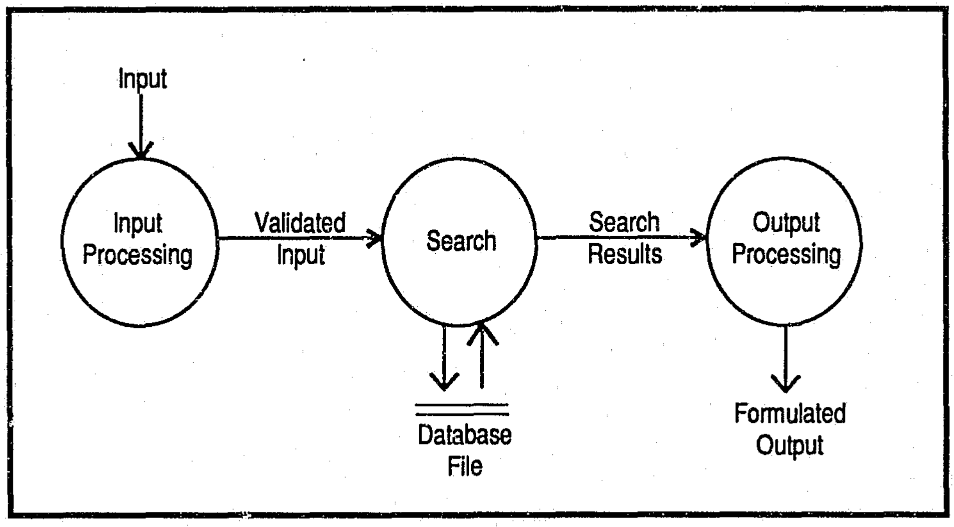


Figure 2: Sample Data Flow Diagram

3.4 Description of Modules

Describe the attributes and requirements of each software module. The requirements should define the function of the module in terms that can be measured and tested, keeping in mind the need to design test data in the later phases of development. Explain how each module relates to the other modules and functions of the system.

Describe the inputs, processing, and outputs of each module. For inputs, include a description of the data, its type, source and bounds. Supply a chart if needed for clarity. For processing requirements, include a discussion of the techniques or algorithms used, as well as the parameters and bounds of the processing. For outputs, include the format of any information or data and its destination. Provide diagrams if required for clarity.

3.5 Design Limitations

Specify limitations imposed on the design by system architecture and development tools. For example, there may be a need to use a different language in some modules or functions, such as a C routine in a Fourth Generation Language Database Management System (4GL DBMS). The methodology used in the design may affect the compatibility of the modules and functions. Especially important is a discussion of the design considerations for ease of maintenance or future modifications and enhancements. Discuss internal features that make testing of the software easier (for example, trace features, temporary files, global memory variables, intermediate printouts, and internally commented code). Finally, if security features are included, discuss their design parameters and reliability.

3.6 Data

Identify the system's data requirements. Files and databases created for or accessed by the proposed software must be described in terms of the data format and structure, content, number of data items, and types of access (for example, single/multiuser or indexed/sequential). Capacity requirements should be specified in terms of the maximum number of records, users and workstations. Include global data items such as tables or site files.

Taken together, the End User Requirements and Systems Specifications form the conceptual understanding of the proposed system. They are, in essence, the blueprint for the design of the system.

3.7 Design Validation Test Criteria

Provide an overview of the types of system testing needed to validate compliance of the final product with the System Specifications. This type of testing is known as Validation Testing. More information on testing will be provided in the Test Specifications.

4.0 Hardware

Specify the computer system upon which the proposed software will operate. Provide an overview of the system's performance standards, design criteria, development requirements and test requirements. This information creates a technical baseline for the development of the system.

4.1 Hardware Components

Provide a brief overview of the components of the existing system, including a description of basic functions and interfaces. A system diagram will help to ensure that the development team has a comprehensive understanding of the operation, flow of information and control in the system. Describe the communications interfaces between the existing system and any other systems. List every component of the system (hardware and software) by manufacturer, model, nomenclature and version. If the list is extensive, put it in an appendix.

4.2 Hardware Specifications

Describe the architecture of the system, including the microprocessor model, and performance specifications (such as MIPS, Mhz, etc.), and reference all other processor related information which might be of use to the development team. Include the architecture and performance of the memory, system bus, display adapter, disk and tape storage devices and their media formats, power supply, Input/Output ports, and keyboard/pointing devices. Also include the operating system, network operating system, or programming language requirements mandated by the system or network itself (such as a specific version of DOS or BASIC). Specify reliability data, such as Mean Time Between Failures and Mean Time Between Repair for each major component in the system. Finally, identify any environmental conditions which may be encountered in the operation of the equipment, such as noise, electrical interference or temperature.

4.3 Hardware Interfaces

Describe hardware interfaces that affect the proposed software, including:

- Manufacturer, model, and speed of the processor;
- System memory size in Kb or Mb, and memory speed in nanoseconds;
- Seek and access times for disk storage devices;
- Format and capacity of storage media, such as tape or diskettes;
- Display attributes such as medium or high resolution color graphics; and
- Special hardware designed to test the software or required as a part of a prototype module (such as a LAN adapter).

4.4 Hardware Operational Procedures

Identify collection and recording procedures necessary to maintain the reliability specifications (for example, the format and procedures for usage and problem logs). Define the minimum operational standards of the system in terms of daily operation, backup procedures, and consumable supplies, along with references to the applicable documentation. Note areas in which an operator could, through error or negligence, cause damage to the system or loss of data.

4.5 Hardware Technical Support

Identify available hardware performance testing procedures or media and memory checking programs which may be used to confirm that the system is operating according to specifications. Identify procedures for maintenance and support of the system, including contact names, phone numbers, hours of available maintenance, location of parts or service facilities and maintenance functions provided by staff.

4.6 Personnel and Training

Specify additional personnel or skills the development team needs to use the hardware. If additional training is required, specify the location, cost and schedule of the required classes.

5.0 Interface Limitations

The components required by the total system environment, such as the choice of hardware, programming languages and the system and user interfaces, force limits on the design. List any design restrictions and describe how they may affect the entire system. For example, if the system must be capable of using a light pen as a primary input device, then the software must create the signals that the pen can interpret and the user interface (the computer screen) must be designed to interact with the pen, using, for example, check-off boxes.

6.0 Appendix

Include such things as flowcharts, tables and diagrams.

Preliminary Project Plan

The Preliminary Project Plan defines the nature and extent of the software development effort, based on the information in the End User Requirements and the System Specifications. The Preliminary Project Plan provides an estimate of personnel and technical resources, schedules and costs necessary to accomplish the work. The critical function of the Preliminary Project Plan is to make an early assessment of the feasibility of the project. It essentially determines whether the project should and can go forward, or whether the nature and scope of the effort, available resources or costs are prohibitive.

The Preliminary Project Plan estimates the scope of the project and the resources needed to fully execute all necessary tasks.

Since the Preliminary Project Plan's purpose is primarily to estimate the scope of the project and the resources needed to fully execute all necessary tasks, and is intended to allow management to determine whether the project is to go ahead, it makes preliminary assessments with limited objectives. If project management determines that the project will go forward, then a more detailed Project Plan will be created, incorporating essential information from the Preliminary Project Plan.

The objectives of the Preliminary Project Plan are to:

- define the scope of the project;
- identify tasks and deliverables;
- identify available resources;
- establish schedules; and
- estimate costs.

Define Scope. The general scope of the project is contained in the End User Requirements and the System Specifications. Based on the functions the end user needs and the preliminary specifications of the system, the development team is able to make preliminary assessments of what level of effort will be necessary to develop a computer solution.

Identify Tasks and Deliverables. For each phase of the development project, tasks and deliverables need to be identified to develop an estimate of the work to be done in the project. Descriptions of tasks and deliverables can be in general terms (such as the need for a data entry module and a report module rather than the shape, content and format of the display screens). See *Table 1: Systems Development Tasks and Deliverables*, page 16.

PHASE	TASK	DELIVERABLE
Planning	Requirements Analysis	Requirements Specifications
	Systems Specifications	Systems Specifications Document
	Preliminary Project Plan	Preliminary Project Plan Document
	Project Plan	Project Plan Document
	Change Control	Problem Identification Form Change Request Form
Design	Preliminary Design	Preliminary Design Document Preliminary User's Manual
	Preliminary Design Walkthrough	System Prototype Program
	Detailed Design	Test Specifications Document
	Detailed Design Walkthrough	Detailed Design Document
Coding and Testing	Coding	
	Coding Walkthrough	
	Unit Testing	Individual Module Code
	Integration Testing	Integrated Module Code
	Validation Testing	Validated System Code
	Site Testing (Beta Testing)	Completed Design Document User's Manual
Operation and Maintenance	Operations Management	
	Maintenance	

Table 1: Systems Development Tasks and Deliverables

Identify Available Resources. There are three basic resources to allocate:

- Human Resources: analysts, programmers, and administrative help;
- Hardware Resources: the computer system and peripheral devices upon which the software operates; and
- Software Resources: the operating system and programming languages.

What may appear to be a simple software program requiring few resources can grow into a complex application, consuming large amounts of hardware, software and personnel resources. Projection of resources is one of the most difficult tasks, because knowledge of all the facets of the design process are usually only learned from experience. Careful monitoring must be conducted and documentation should be kept on all development work to provide a historical record of the development effort. This record will form the basis for estimating resource utilization in subsequent system design projects.

Establish Schedules. A realistic estimate of the time necessary to complete the project must be established. In addition, a proposed schedule which assigns all of the resources to be used in the project must be created.

Project scheduling specifies realistic milestones, or the completion of a task or a sequence of tasks. For example, completion of the Project Plan is a milestone. Each milestone should include a deliverable, which is something tangible that may be examined, reviewed, tested or demonstrated, such as a document, a schedule or a software program.

The two most common errors made in project scheduling are assigning timelines without considering resource requirements for the specific task (such as the number of required personnel or tools) and setting unrealistic completion dates. Understanding the nature and scope of the tasks, and knowing the capabilities of the personnel who will be performing the tasks, greatly assists estimating schedules.

Estimate Costs. A number of cost-projection techniques exist for assigning accurate costs for a software development project. The most common are the Lines-of-Code and the Task-Costing techniques. Both methods depend on some familiarity with the development process and base their estimates on the historical records of previous development projects. Barry W. Boehm's *Software Engineering Economics* is a highly regarded work on cost estimation in software development.[1] (See the *Recommended Reading List* in the Appendix, page 58.)

If cost estimates for the project are higher than the amount anticipated, consider "off-the-shelf" routines, modules or development support programs to reduce the development effort. When purchasing "off-the-shelf" software, consider

[1]B. Boehm, *Software Engineering Economics* (Englewood Cliffs: Prentice Hall, 1981).

issues such as interfaces, installed user base, maintenance and enhancement policies, source code availability, benchmarks, training and evaluation of potential vendors.

Once the Preliminary Project Plan has been prepared, management personnel and the end user should review it for feasibility, scheduling, costs and resource requirements. These requirements, resources and costs will be reevaluated and refined as other planning documents are produced. At the conclusion of the planning process, the Preliminary Project Plan will become the Project Plan.

The information contained within the System Specifications, along with the resources defined in the Project Plan, will form the basis for all of the cost and schedule estimates made in the next phase.

Preliminary Project Plan

Preliminary Project Plan Document Format

1.0 Project Overview

The Preliminary Project Plan begins with an overview of the project, identifying the end user of the software, the background of the request and the intended purpose of the software. Includes major assumptions or restrictions identified in the Requirements Analysis.

2.0 Functional Overview

Describe the major functions of the proposed software in a few short paragraphs. Comment on unique functions of a proposed module. A diagram showing the interaction of the various software modules is recommended.

3.0 Tasks and Deliverables Overview

Summarize the software development tasks and the deliverables required by the project. Define each task in sufficient detail to provide for a baseline concept of the proposed system and to allow for accurate cost and time scheduling estimates. Later in the project all basic tasks will be refined, specifying subtasks for each major task.

Provide an outline of the procedures necessary to prepare the required project documents. Provide an outline of the training requirements, including additional training that the development team may need and training that the end users will need upon completion of the project.

4.0 Resource and Resource Costs Overview

To estimate costs, identify the available resources allocated to each task. A resource is anything that can be used to accomplish the planning, development, testing or maintenance tasks, such as personnel, hardware and software. For personnel resources, include the number of people required for each task, the skills required, and the percentage of effort required by each person (for example, 50 percent use of programmer #1 for 15 days).

Identify each piece of hardware to be used in the planning and development phases, along with the required duration of use and associated costs. Finally, identify support or utility software required for use in the administration, management or development and include cost estimates. The estimate of the project costs must be made known to the user.

5.0 Scheduling Overview

A chart or table (such as the *Systems Development Tasks and Deliverables* table on page 16) can be used as the basis of scheduling tasks, deliverables and resources. Include dates of activities which might affect the time schedule. Include hardware or software delivery dates, limitations on the availability of the development staff (such as holidays or maternity leave), and the projected end date of the development project.

6.0 Personnel Overview

Provide a diagram of the proposed positions and hierarchy of personnel available for the project. Show reporting structure, considering job assignments and responsibilities. (See *Figure 3: Sample Personnel Structure Overview.*)

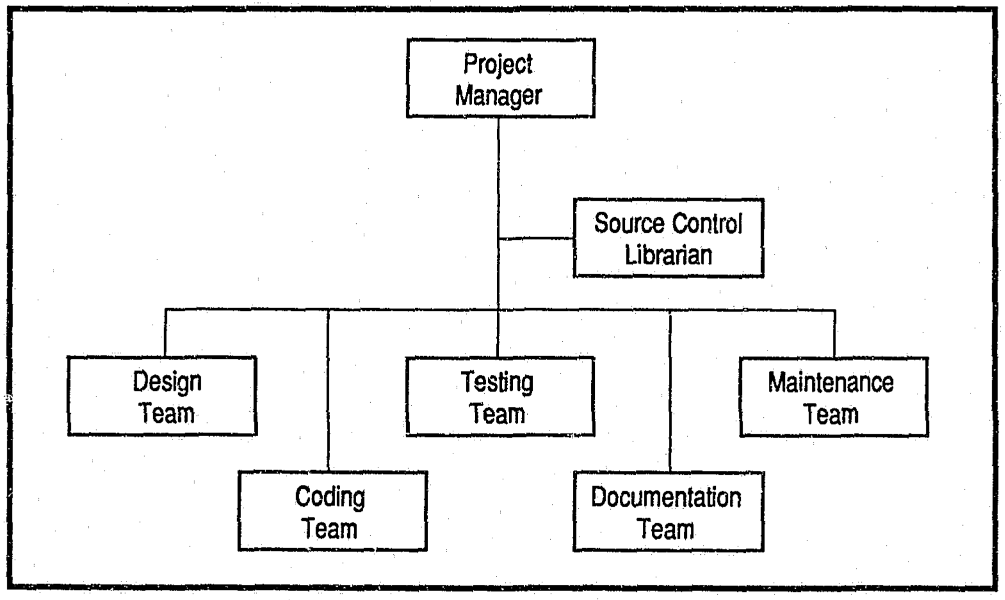


Figure 3: Sample Personnel Structure Overview

Project Plan

The Project Plan is the strategy document for the project, describing how the project manager will control and execute all work necessary to produce the proposed system. As such, the Project Plan must integrate all phases of the project, assign realistic schedules, estimate all technical and personnel resources, and estimate all costs. Clear definitions must be provided for *what* is to be done, *when* it is to be done, by *whom* and at what *cost*. Most important, however, is *how* it will be done — the methods for ensuring that all work is executed in a structured and controlled manner. A successful Project Plan, then, accomplishes at least the following objectives:

- Provides management, technical personnel and the end user with an overview of the nature and scope of the project;
- Provides management with a methodology for keeping all project activities visible and under control at all times;
- Defines a baseline system configuration, which is understood and approved by management, the technical development team and the end user for whom the system is being created;
- Contains the tasks and deliverables that must be accomplished, with associated milestones to monitor progress;
- Clearly defines acceptance criteria for all deliverables to the end user during and at the completion of the project, including software, documentation and essential services;
- Lists key personnel in administration, management and technical development and carefully defines their roles and responsibilities;
- Contains detailed budgets and schedules by task and by person; and
- Establishes all critical documents that must be produced to define work to be done, record progress, and account for necessary changes in the design of the system.

The Project Plan need not be a lengthy document. Its importance resides in the fact that it precisely defines the work to be done and establishes a methodology that ensures that the work will be successful. Moreover, the Project Plan functions as an agreed-upon blueprint for all parties involved. Management, technical personnel and the end user agree that the capabilities of the system defined in the plan can be referenced in the End User Requirements and the Systems Specifications, and that all deliverables meet all End User Requirements. It is crucial to incorporate acceptance criteria in the plan to assure complete agreement on what is to be delivered to the customer at the conclusion of the project. The Project Plan, ultimately, functions as both an enabling document and an agreement of the definition of the product.

Project Plan

Project Plan Document Format

1.0 Project Overview

A detailed overview of the project includes the identity of the software end user, the background of the request, the intended purpose of the software, and major assumptions or restrictions used to develop the project plan.

2.0 Major Functions and Bounds

Briefly describe the major functions of the proposed software. This will place bounds on the project during the development process. The use of a diagram showing the interaction of the various modules and submodules is recommended. Briefly describe unique subfunctions of the modules.

The Project Plan defines the strategy for the project. It must integrate all phases of the project, assign realistic schedules, estimate all technical and personnel resources, and estimate all costs.

Identify the computer and operating system to provide a hardware and operating system baseline for the project. Include a system-level block diagram of the system hardware and software functions. List performance requirements and characteristics, including memory requirements and limits, minimum hardware requirements, and any special requirements (such as a mouse, or optical scanner). Note any features or functions required in the software or system design to ensure the reliability of the system and software (such as full hardware redundancy, fault tolerance, or a software roll-back function). Include software interfaces to other devices (such as the utility software to operate a tape drive) and the method of interface.

3.0 Tasks and Deliverables

Summarize the tasks and the deliverables required by the project. Define each task in sufficient detail to allow for accurate cost and time scheduling estimates. Later in the project these basic tasks will be refined, specifying subtasks for each major step. After the major tasks are defined, list the deliverables associated with each task. (See *Table 1: Systems Development Tasks and Deliverables*, page 16).

Provide an outline of the procedures and personnel responsibilities involved in the preparation and distribution of the project documents. Include the personnel responsible for creating the documentation required in each phase of the project, including typing support, editing, document reproduction and distribution.

Provide an outline of training requirements and personnel responsibilities. Include additional training for the development team in the applicable programming language, use of the development hardware and operating system, documentation procedures, project management or other related skills. Define the training requirements for the end users upon project completion, such as installation, operation and maintenance of the system.

Provide an outline of the system's installation requirements. Include the schedule for installation and responsibilities of the end user, management and technical teams. Other requirements include the equipment and power required, personnel schedules and system security. Define procedures to be used during the transition, or "cut-over," from the existing system to the new system. Consider issues such as parallel operation, conversion criteria, responsibilities for the decision to make the conversion and a fall back position should the new system not work according to plan. Specify the time-frame in which the end user can expect operational support and enhancements.

4.0 Resources and Resource Costs

To estimate costs or schedules, identify the available resources for each task. (A resource is anything that can be used to accomplish the planning, development, testing, or maintenance tasks, such as personnel, hardware and software.)

Estimate personnel required and associated costs. Identify the number of people required for each task, the skills required, and the amount of time required for each person. Project management tools such as Gantt charts are helpful in determining personnel costs over the duration of the project. (See *Figure 4: Sample Gantt Chart.*) Personnel costs include programmers, analysts, technicians, and management personnel, as well as non-technical staff such as secretarial, editorial, and administrative personnel. Labor costs for each task by person should be accounted for in a tabular format, prepared in conjunction with the personnel requirements. Identify costs applicable to personnel, such as training, travel, per diem and vacations.

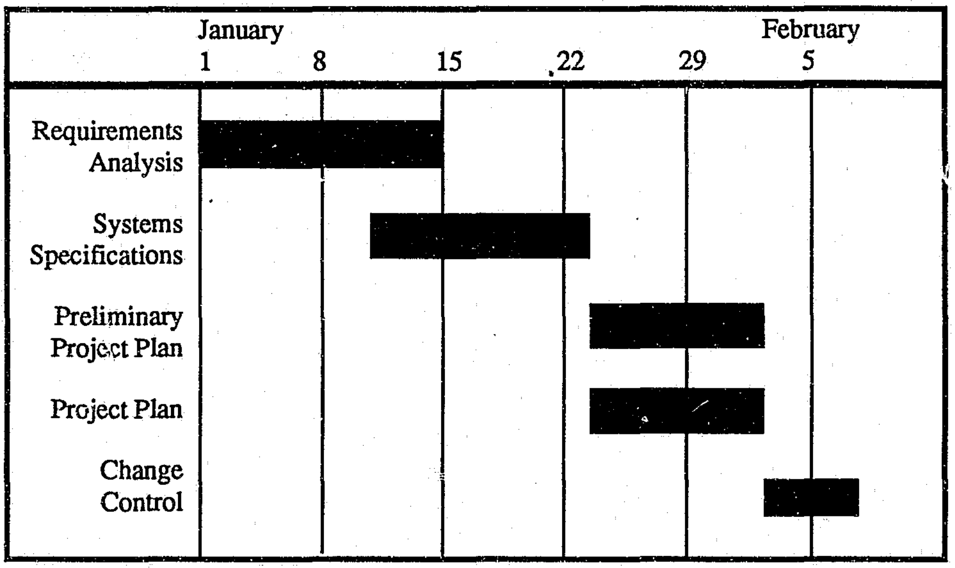


Figure 4: Sample Gantt Chart

Estimate hardware costs. Each piece of hardware should be identified by type, quantity, and duration of use. Estimate project support cost, such as photocopying, word processing, administrative use of computers, and the hourly cost for computer time by the development and coding staff. A Gantt chart can show the hardware utilization and associated costs on a monthly basis over the duration of the project.

Estimate software costs, such as operating systems, development languages, test programs and utilities, and application packages for administrative tasks or document preparation.

5.0 Scheduling

Provide a table or chart depicting the schedule for all tasks, deliverables, and resources. Include dates of activities that might affect the schedule, such as hardware delivery, availability dates, limitations on the availability of the development staff (such as holidays or maternity leave), and limits on software resources.

Project management analysis techniques, such as the Program Evaluation Review Technique (PERT) or Critical Path Management (available on micro-computer software packages), are valuable assets to the scheduling effort. Such automated programs allow for the integration of all resources, tasks, and deliverables, along with their respective timelines, critical paths and milestones. (See *Figure 5: Sample PERT Chart.*)

Project management software also facilitates revisions, allows the operator to assign costs and availabilities to each resource, and maintains ongoing cost analysis.

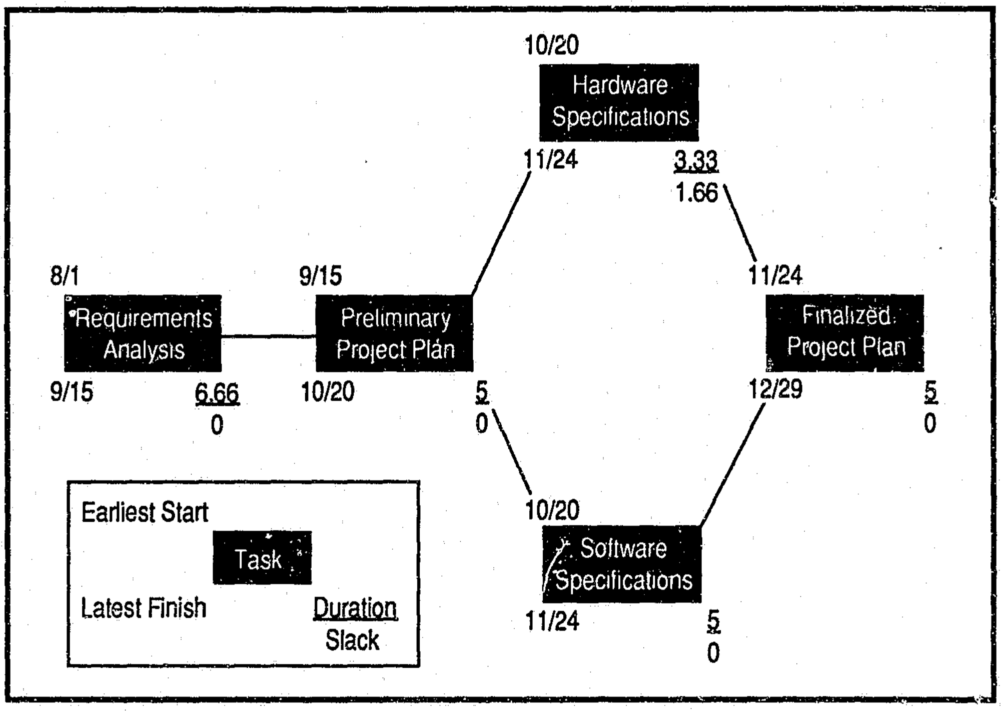


Figure 5: Sample PERT Chart

6.0 Personnel Structure

Provide a graphic presentation of the positions and hierarchy of personnel available for the project. This should be a more detailed version of the diagram constructed in the Preliminary Project Plan. (See *Figure 6: Expanded Personnel Structure.)*

While there is no single approach to organizing personnel in a development effort, a team-oriented approach is commonly employed. This approach

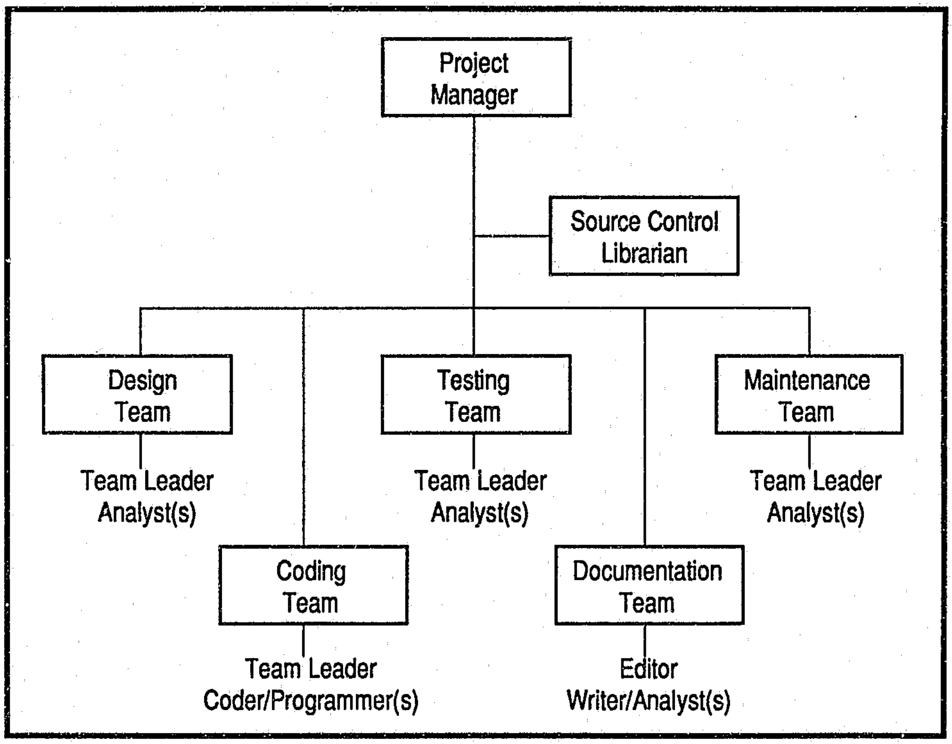


Figure 6: Expanded Personnel Structure

allows the project activities to be split among the teams and performed in parallel. Project management is easier under a team approach and there is less reliance on a single staff member.

Typically, development teams are organized with a Team Leader, an assistant, and other technical personnel. The Team Leader is usually an experienced software professional who has final responsibility for the efforts of the team. A Team Leader performs the top-level design and programming activities with the other team members, participates in all reviews, and acts as liaison with the project manager and other teams.

In development efforts with limited personnel, the same individuals may have to staff multiple positions. There is, however, a rule of thumb that "designers should not code and coders should not test." It is an admonition both against spreading skills too thin and not having personnel test and evaluate their own work. Personnel resources and skills should be critically evaluated in the planning phase of the project. Consults may be required if there is not a sufficient base of skills in-house.

A Documentation Team composed of an experienced editor and other writers/analysts are needed to assist the development team in preparing the User's Manual and associated documentation. It is important that the Documentation Team be involved from the outset in the planning and writing of all documentation.

The Project Manager directs the activities of each of the Team Leaders. The Source Control Librarian is usually a systems analyst who reports directly to the Project Manager and is responsible for all the project documentation.

The Team Leaders and other key personnel will be required to assume additional responsibilities by serving on the following committees: Problem Information Forms Review, Change Control Board, Preliminary Design Review and Detailed Design Review. Additional personnel will also sit on these committees. (See *Figure 7: Sample Committee Assignments.*)

	Maintenance Team Leader	Programmer/Analyst	Design Team Leader	Coding Team Leader	Testing Team Leader	Documentation Editor	Project Manager
Problem Information Form Review Committee	●	●	●	●	●		
Change Control Board	●		●	●	●	●	●
Preliminary Design Review Committee	●		●	●	●	●	●
Detailed Design Review Committee	●		●	●	●	●	●

Figure 7: Sample Committee Assignments

Change Control

Change control is a formal mechanism for the identification, evaluation and approval of all changes in the design of the system.

Design changes requested after the design documents have been approved are an inevitable part of the software development process. Changes are requested for a variety of reasons. End users learn more about computers and the proposed system as the process evolves and inevitably desire modifications of the original design. The system coders request changes to accommodate problems that surface only during the actual development process. Changes are also inevitable as a result of testing, when errors are encountered in trying to integrate all of the modules into a functioning system. Other reasons for change are design deficiencies, test procedure errors, incompatible interfaces, errors in connectivity to support software such as test drivers or compilers, and operating system problems. It is simply a fact that even in the best planned and designed system, neither the end user nor the development team can foresee all problems that will arise.

The issue is not that changes should be avoided or disallowed, but that changes should be controlled. Accordingly, change control is a formal mechanism for the identification, evaluation and approval of all changes in the design of the system. As a critical function of the overall management of the software development process, rigorous change control ensures that all modifications to the system are visible to management, technicians and the end user. Management must be aware of all substantive changes to control the development effort. Changes can have a significant impact on budgets, schedules and resources. Most important, however, is that if changes are unauthorized and undetected, then ultimately there is a danger that the system may not meet the requirements of the end user.

When change is necessary, it is imperative that it be handled in a controlled manner and that the development team and the end user are in complete agreement about the change. For example, in the coding process, the coder finds a functional problem with the configuration. He therefore recommends a design change to rectify the problem. The proposed change, however, will result in a modification of the original design as approved by the developer and the end user. At this point, the end user must be made aware of the nature of the modification and his approval must be obtained. In similar fashion, the project manager must be made aware of any change requested by the end user, since it will affect budgets, schedules and resources.

It is critical that responsibility be assumed by the party requesting changes in the design of the system. That is why all changes must be controlled, visible and traceable by everyone involved in the development effort and why all changes should be in writing and signed. Historical records should be kept on all changes.

Change Control Board (CCB)

The establishment of a Change Control Board ensures that all changes will be made in a controlled manner.

Formality is brought to the process through the mechanism of a Change Control Board (CCB) and through the use of Problem Information Forms (PIFs) and Change Requests Forms (CRFs). The Change Control Board and the PIFs and CRFs also make the change process visible to management, thereby ensuring management control of the project. The CCB is empowered with the authority to accept or reject change. The composition and size of the CCB depends on the nature of the development project and the size of the organization undertaking the project. It is important that the end user as well as the developer has representation on the CCB. Responsibilities of the CCB include:

- Reviewing and approving all requests for design changes;
- Assigning priorities to the requests;
- Evaluating the technical, administrative and managerial effects of the changes requested;
- Determining the most likely cause for the error;
- Establishing an estimate of the time required to effect the correction/change;

- Evaluating the effect implementing the proposed correction/change will have on the overall project schedule;
- Establishing new validation criteria for the changes in the design;
- Following up on each CRF to verify the CCB's estimates; and
- Ensuring that all changes are documented and placed in the Source Control Library.

Problem Information Forms and Change Requests Forms

Any errors or change requests should be recorded on the appropriate form and forwarded to the Change Control Board.

Problem Information Forms (PIFs) and Change Request Forms (CRFs) (see Figures 7-10) are the only means for reporting errors and requesting changes to the design. When a member of the development team identifies a problem or desires an enhancement, he prepares one of the forms and submits it to the Change Control Board (CCB).

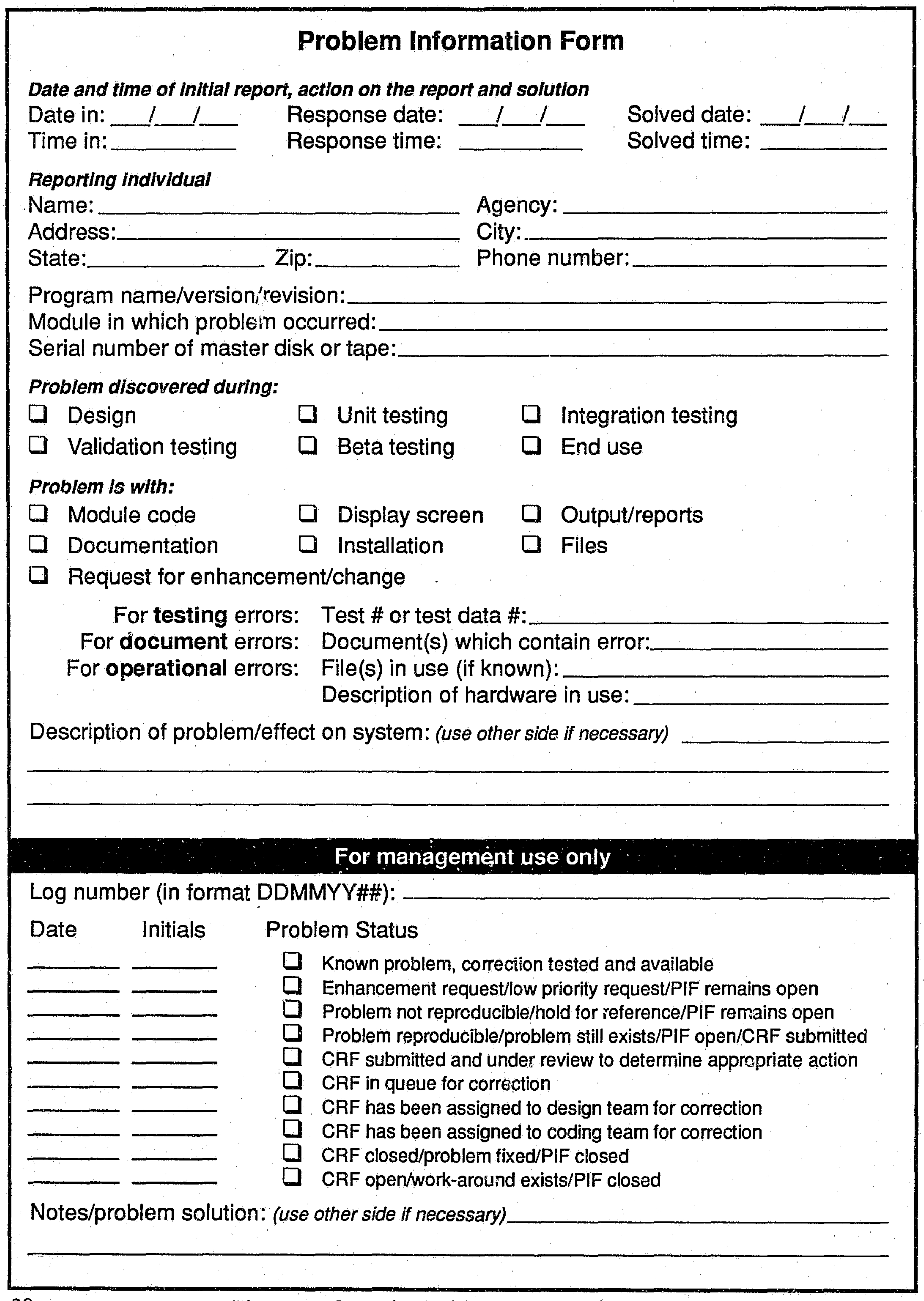
In completing a PIF, include all available information on the problem, a description of the hardware environment, the actions which preceded the problem, and the version and revision level of the software in question. The PIF is then forwarded to the Team Manager.

The Team Manager completes the top section of the CRF based on the information in the PIF. A copy of the PIF is dated and placed in the Source Control Library (SCL) and the original PIF is attached to the CRF and forwarded to the CCB. A CRF is the only means of obtaining change approval from the CCB, which has sole authority to change the design of the program.

A proposed change must first be assessed to determine the cost of implementation, scheduling repercussions, and performance effects upon the system. The CCB then assigns a priority and a "change classification" to the CRF. The change classification is used to identify whether the change is to correct a flaw, to create an enhancement, or perform a port (the process of converting the software to run on a different hardware environment). After classification, the CCB will estimate the required resources to perform the change and, if necessary, will schedule the design or maintenance team to make the change. The CCB also assigns a subclass to the CRF to identify whether the error requires maintenance scheduling, has a "work-around" (a means to circumvent its occurrence), or whether a remedy for the error already exists. The CCB will then indicate on the CRF the action taken and return a copy of the (CRF) and (PIF) to the Team Manager who submitted them.

In addition to providing a mechanism for authorizing and controlling requested changes in the development process, the benefit of using PIFs, CRFs and a CCB is that groups of related errors may be corrected at the same time. By performing changes in groups, small changes which might otherwise be ignored or delayed may be performed at the same time as larger changes.

Changes and enhancements requested during the Operational and Maintenance Phase also should be controlled by the Change Control Process.



Problem Information Form

Date and time of initial report, action on the report and solution

Date in: ___/___/___ Response date: ___/___/___ Solved date: ___/___/___
Time in: __________ Response time: __________ Solved time: __________

Reporting individual

Name: ____________________ Agency: ____________________
Address: ____________________ City: ____________________
State: __________ Zip: __________ Phone number: ____________________

Program name/version/revision: ____________________
Module in which problem occurred: ____________________
Serial number of master disk or tape: ____________________

Problem discovered during:

☐ Design	☐ Unit testing	☐ Integration testing
☐ Validation testing	☐ Beta testing	☐ End use

Problem is with:

☐ Module code	☐ Display screen	☐ Output/reports
☐ Documentation	☐ Installation	☐ Files
☐ Request for enhancement/change		

For **testing** errors: Test # or test data #: ____________________
For **document** errors: Document(s) which contain error: ____________________
For **operational** errors: File(s) in use (if known): ____________________
Description of hardware in use: ____________________

Description of problem/effect on system: *(use other side if necessary)* ____________________

For management use only

Log number (in format DDMMYY##): ____________________

Date	Initials	Problem Status
______	______	☐ Known problem, correction tested and available
______	______	☐ Enhancement request/low priority request/PIF remains open
______	______	☐ Problem not reproducible/hold for reference/PIF remains open
______	______	☐ Problem reproducible/problem still exists/PIF open/CRF submitted
______	______	☐ CRF submitted and under review to determine appropriate action
______	______	☐ CRF in queue for correction
______	______	☐ CRF has been assigned to design team for correction
______	______	☐ CRF has been assigned to coding team for correction
______	______	☐ CRF closed/problem fixed/PIF closed
______	______	☐ CRF open/work-around exists/PIF closed

Notes/problem solution: *(use other side if necessary)* ____________________

Figure 7: Sample Problem Information Form

Date and time of initial report, action on the report and solution:

This section is used to record the date and time when the PIF was submitted; responded to; and finally solved. In some cases, the response date may not be the same date as the submittal date, and the solved date may be much later (in the case of an open PIF).

Reporting individual:

This section is used to record the pertinent information on the reporting individual in case the CCB needs to contact them for further information.

Program name/version/revision:

Self explanatory.

Module in which problem occurred:

Self explanatory.

Serial number of master disk or tape:

Self explanatory.

Problem discovered during:

This section is used to designate at which step in the development process the problem was discovered. Generally this information is saved for historical and statistical purposes, and to give the project management an indication of problem trends developing from any particular step in the development process.

Problem is with:

This section gives a quick reference to the type of problem discovered, which allows the CCB to generally categorize the PIF. Multiple selections are allowed.

For testing errors:

Test # or test data #:	Self explanatory.

For document errors:

Document(s) which contain error(s):	Self explanatory.

For operational errors:

File(s) in use (if known):	Self explanatory.
Description of hardware in use:	Self explanatory.

Description of problem/effect on system:

This section is used to give a narrative description of the problem, and to provide as much specific detail as possible to help management determine the nature of the problem. The problem description will also be used by the CCB to assign a priority to the CRF generated by the PIF. Additional space is available on the reverse side of the form.

For management use only:

This section of the form is completed by the project management personnel responsible for design and maintenance.

Log number:

This is a unique identifier used for control and reference purposes. Usually composed of a numeric code such as: 15068803 where the structure DDMMYY## is used to represent a day-month-year-sequence. Thus: 15 is the 15th day of the month, 06 is June, 88 is 1988, 03 means the 3rd PIF of the day.

Date:

Date of activity.

Initials:

Initials of responsible manager.

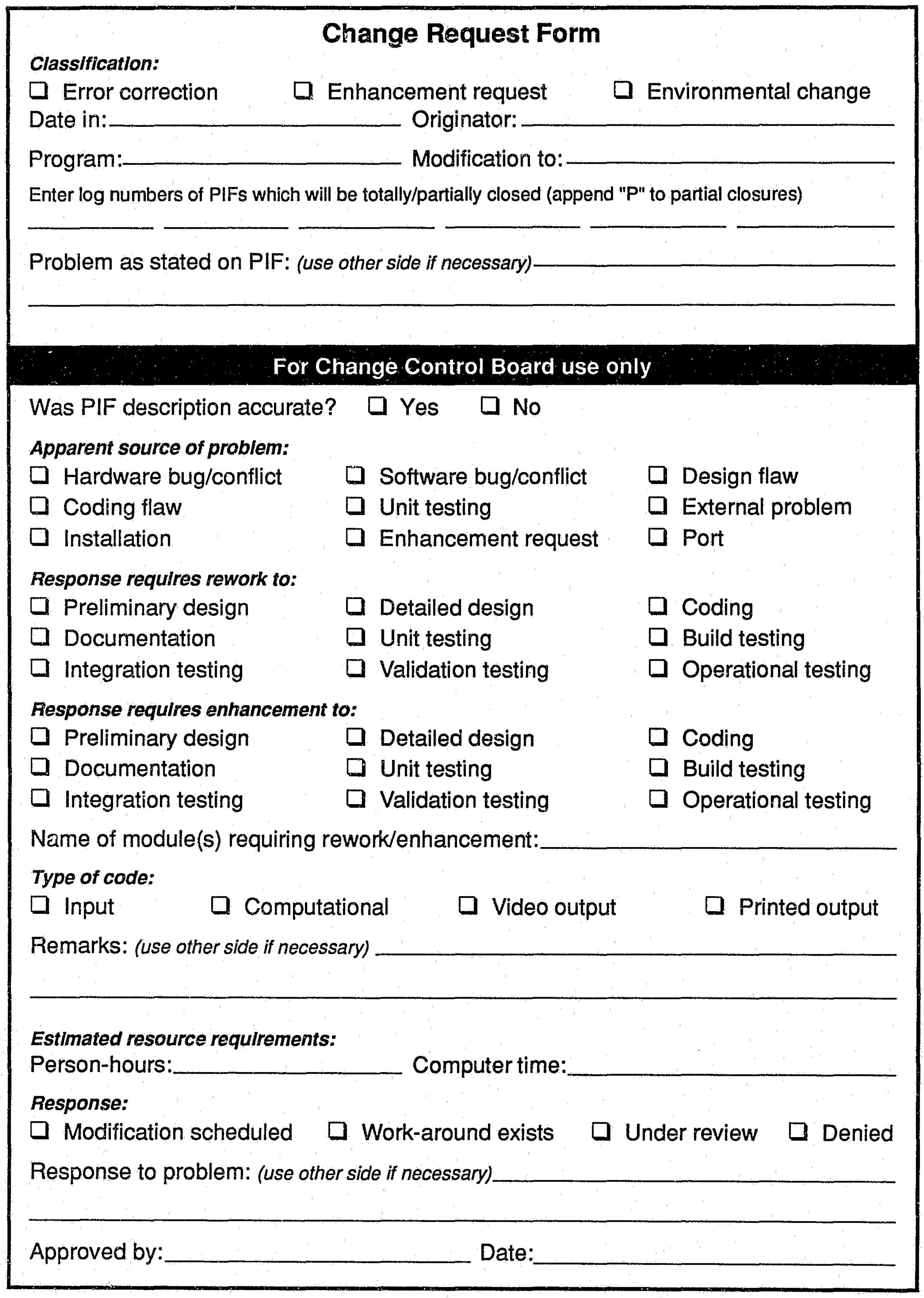
Problem status:

This section provides a brief status of the PIF and any follow-up actions generated by the PIF. Blocks are checked as appropriate. Management is responsible for informing the CCB of the status of each PIF. Closed PIFs designate the problem has been resolved in some fashion. Open PIFs mean that activity is still in progress on the problem. Management must ensure that overall project scheduling is adjusted based upon these actions if necessary.

Notes/problem solution:

This section is used to record any notes about the problem, the actions of the CCB, or a short statement on the resolution of the problem.

Figure 8: Explanation of Sample Problem Information Form



Change Request Form

Classification:

☐ Error correction ☐ Enhancement request ☐ Environmental change

Date in: ____________ Originator: ____________

Program: ____________ Modification to: ____________

Enter log numbers of PIFs which will be totally/partially closed (append "P" to partial closures)

______ ______ ______ ______ ______ ______

Problem as stated on PIF: *(use other side if necessary)* ____________

____

For Change Control Board use only

Was PIF description accurate? ☐ Yes ☐ No

Apparent source of problem:

☐ Hardware bug/conflict	☐ Software bug/conflict	☐ Design flaw
☐ Coding flaw	☐ Unit testing	☐ External problem
☐ Installation	☐ Enhancement request	☐ Port

Response requires rework to:

☐ Preliminary design	☐ Detailed design	☐ Coding
☐ Documentation	☐ Unit testing	☐ Build testing
☐ Integration testing	☐ Validation testing	☐ Operational testing

Response requires enhancement to:

☐ Preliminary design	☐ Detailed design	☐ Coding
☐ Documentation	☐ Unit testing	☐ Build testing
☐ Integration testing	☐ Validation testing	☐ Operational testing

Name of module(s) requiring rework/enhancement: ____________

Type of code:

☐ Input ☐ Computational ☐ Video output ☐ Printed output

Remarks: *(use other side if necessary)* ____________

____

Estimated resource requirements:

Person-hours: ____________ Computer time: ____________

Response:

☐ Modification scheduled ☐ Work-around exists ☐ Under review ☐ Denied

Response to problem: *(use other side if necessary)* ____________

____

Approved by: ____________ Date: ____________

Figure 9: Sample Change Request Form

Classification:
This section is used by design/maintenance supervisor to designate the type of change requested.

Date in:
This is the date of submission of the CRF.

Originator:
This lists who submitted the CRF in order to allow the CCB to contact the originator in the event more detail is required

Program/module:
Name of program/module for which change is requested.

Log numbers:
This section is used to list the Log numbers of all the PIFs that will be closed or partially closed when the Change Request is completed. If a PIF will only be partially closed, a "P" is appended to the Log number (ex: 06158803P).

Problem as stated on PIF:
This section is used to provide a narrative of the problem/enhancement/ environmental change in sufficient detail so the CCB can gain a clear understanding of the request.

Was PIF description accurate?
Self explanatory.

Apparent source of problem:
This section is used to specify the type of problem or request.

Response requires rework to:
This section is used to specify all of the steps which will require revision in order to perform the requested modification to the program. All the applicable steps are checked as necessary.

Response requires enhancement to:
This section is used to specify all the steps which will require modification in order to perform the requested enhancement to the program. All the applicable steps are checked as necessary.

Name of module(s) requiring rework/enhancement:
Self explanatory.

Type of code:
This section is used to indicate the type of program code to be modified in order to assist in evaluating the time required for the modification(s).

Remarks:
This section is used if the change or enhancement can be satisfied with an explanation (i.e. a notation that the problem is not a bug, but a feature), or to note that the problem was not reproducible by the CCB.

Estimated resource requirements:
This section is used to specify the estimated person and machine hours required in order to effect the change(s) or enhancement(s).

Response:
This section is used to specify the response to the CRF as determined by the CCB.

Response to problem:
This section is used to provide a narrative of the response as determined by the CCB.

Approved by/date:
Signature of the senior member of the CCB who is responsible to the Project Manager.

Figure 10: Explanation of Sample Change Request Form

Source Control Library

As the design process advances, the documentation associated with each software module seems to increase exponentially. Besides the End User Requirements, System Specifications, design notes, review notes and source listings, there are volumes of documentation created on a daily basis. To maintain control over the amount of documentation generated, a Source Control Library (SCL) is created. Normally initiated at the completion of the Preliminary Design Review, the SCL is the source for all of the development documentation for each of the software modules.

The Source Control Library contains all documentation associated with each software module.

The SCL is a group of file folders or binders, stored in an easily accessible, central location. Each folder within the SCL documents a single module and has a cover sheet identifying the module's function and documenting the current status of the module. (See *Figure 11: Sample SCL Folder Cover Sheet* on page 34). The cover sheet also typically includes the name of the module, the name of the programmer responsible for the module, a reference to the paragraph in the Design document that defines the module, as well as the start and completion dates (both scheduled and actual) for the Coding, Unit Testing and Integration Testing of a module. There is also a line reserved for the initials of the person responsible for each step in the coding and testing process through final approval of the module. The SCL folder cover sheet provides a historical record of the amount of time spent on each module.

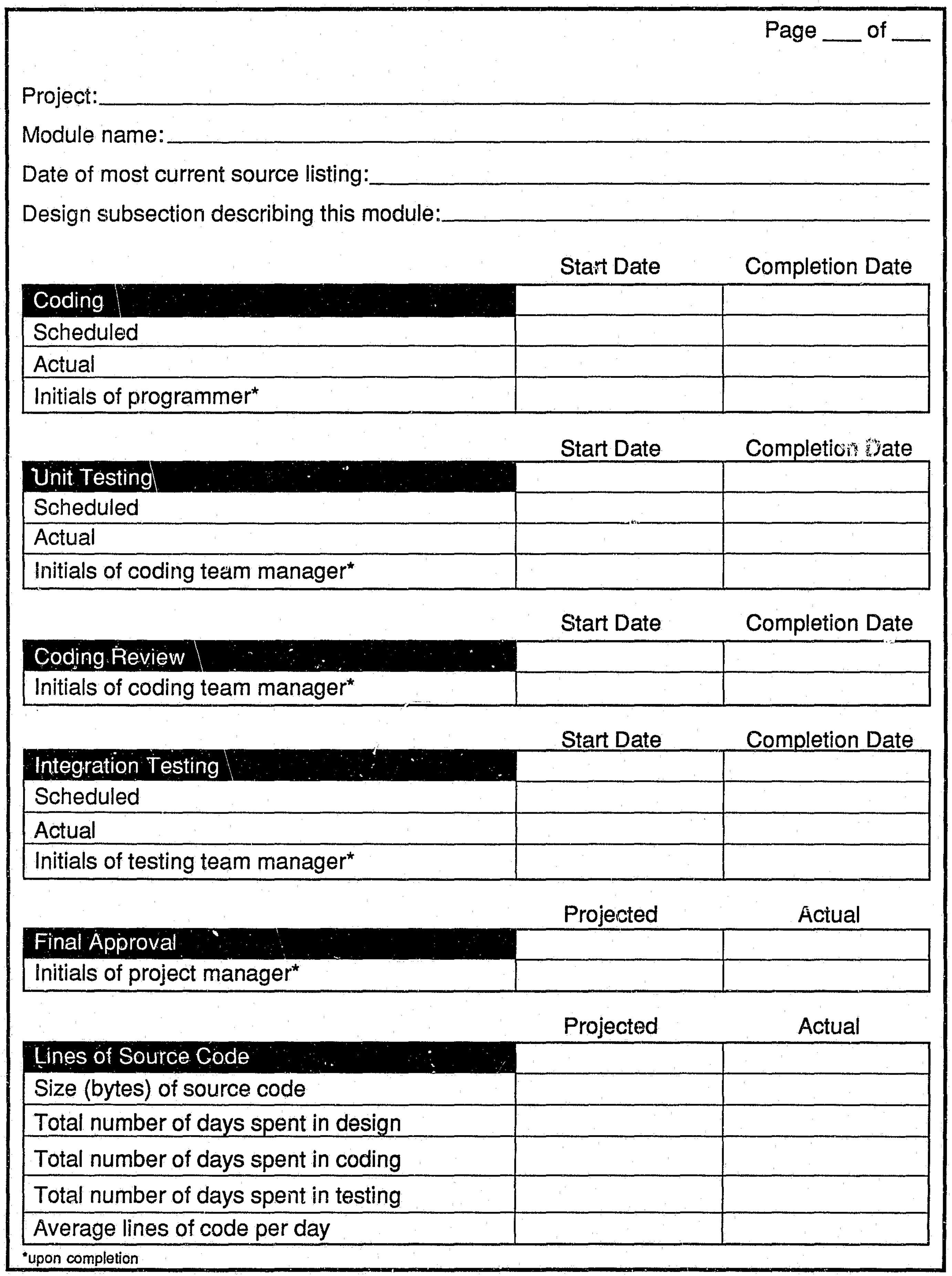
A SCL folder has tabbed dividers separating, at a minimum, the following materials:

- A copy of the Preliminary Design specifications of the module;
- A copy of the Detailed Design description of the module;
- A copy of the Test Specifications relating to the module;
- A chronological file of source code listings, output samples, programmers' notes, and all other documentation generated by the programmers and testers during the development of the module;
- A copy of the first "clean" or error-free source listing;
- An annotated copy of the test results from the Unit Testing;
- A copy of the validated source listing of the module;
- A copy of all Problem Information Forms submitted which references the module; and
- A copy of all Change Request Forms submitted which reference the module.

Source Control Librarian

A systems analyst on the team, known as the Source Control Librarian, is assigned the task of maintaining the Source Control Library, and is responsible for all documentation produced. This includes "failed" runs and "bad" or "buggy" source code. "Bad code" is important historical information about programming errors, design flaws, and lines of code. The Source Control Librarian should develop and disseminate periodic reports on the complete contents of the library, including all new additions, deletions and changes approved by the Change Control Board.

An essential part of the project control process, the Source Control Librarian assists management in keeping all documents accurate and visible, and all changes in the baseline configuration traceable.



Page ___ of ___

Project: ______________________________

Module name: ______________________________

Date of most current source listing: ______________________________

Design subsection describing this module: ______________________________

	Start Date	Completion Date
Coding		
Scheduled		
Actual		
Initials of programmer*		

	Start Date	Completion Date
Unit Testing		
Scheduled		
Actual		
Initials of coding team manager*		

	Start Date	Completion Date
Coding Review		
Initials of coding team manager*		

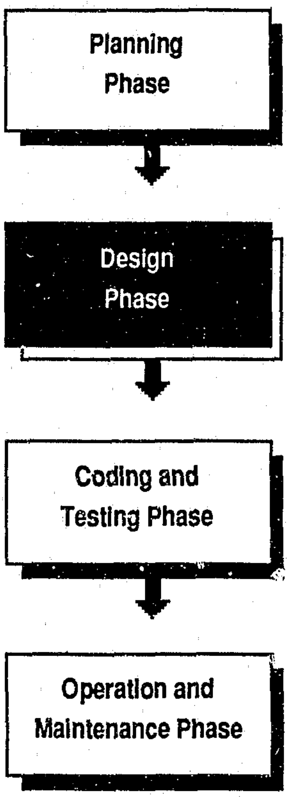
	Start Date	Completion Date
Integration Testing		
Scheduled		
Actual		
Initials of testing team manager*		

	Projected	Actual
Final Approval		
Initials of project manager*		

	Projected	Actual
Lines of Source Code		
Size (bytes) of source code		
Total number of days spent in design		
Total number of days spent in coding		
Total number of days spent in testing		
Average lines of code per day		

*upon completion

Figure 11: Sample Source Control Library Folder Cover Sheet



Design Phase

The purpose of the Design Phase is to create a detailed software design, using the requirements and specifications established in the Planning Phase. The Design Phase is divided into two stages: the Preliminary Design and the Detailed Design. In the Preliminary Design stage the designers specify the overall function and modular structure of the design and create a prototype of the program. A Preliminary Design Walkthrough evaluates the feasibility of the proposed design approach and produces the System Prototype. In the Detailed Design Phase, a design is created that defines the functions and internal procedures of each module in the system.

It is important to note that the testing procedures specific to each module must be established while the module is being designed, even though the procedures will not be applied until after the module is coded in the Coding and Testing Phase. Thus, the Test Specifications Document, which determines the testing specifications for each module, is created during the Detailed Design stage. This document directs the order, schedule and testing methodology which will occur in the Coding and Testing Phase. Finally, the Detailed Design Walkthrough reviews the logic, structure and interfaces of a module prior to coding.

Preliminary Design

The Preliminary Design provides management with information needed to determine whether the design is responsive to the end user's requirements and whether the design can be implemented, given the resource and cost estimates in the Project Plan.

The Preliminary Design is not the actual software program, but a blueprint for the coders to follow in the next phase of the project. It is a "first cut" at the design. Building upon the global concept described in the System Specifications, the designer creates an architecture for the proposed system which contains detailed solutions to the critical requirements of the system functions. The Preliminary Design describes the overall structure of the system but does not provide details of the internal workings of the modules. The Preliminary Design also describes the flow of the data and control of that flow through the structure. It defines the data elements and file access methodology.

The Preliminary Design should include a diagram of the overall system structure, the flow of the data within the structure a[illegible] [illegible] control of the data flow.

For each module that defines a function, the design provides the bounds on the module's input, processing, output, and intermodule data flow. The designer is not concerned with programming details, but with design characteristics that will allow the final coded product to perform correctly and efficiently. Hallmarks of a well designed system are ease of use, reliability and low maintenance.

At this stage, it is important to realize that the design is an idea in the mind of the designer and, as such, represents only one possible solution. Moreover, the designer must make choices among various design methods and hardware and software tools to create a computer solution. The Preliminary Design, therefore, must be subjected to a series of reviews that will compare the proposed design to other alternatives and will attempt to detect conceptual flaws in the design before coding begins. This review process is called the Preliminary Design Walkthrough.

A structured approach to software development generally requires that more time be spent on designing the software than on the coding process. Time spent on design can reduce costs in the actual production of the software program. Careful planning lessens the number of problems caused by design changes during coding and testing. It also controls cascading errors (errors caused by the correction of another error in another part the program) and greatly reduces the number of procedural errors and operational defects, both of which require additional redesign and recoding.

The Preliminary Design generates the Preliminary Design Document. This contains the blueprint for the proposed system and defines the procedures for coding, reviewing and testing the software. After the Preliminary Design Document is approved, it becomes the Design Document, specifying the internal structure of the individual modules and the programming logic needed to allow the coders to write the programs.

Preliminary Design

Preliminary Design Document Format

1.0 System Overview

Provide an overview of the software design. Identify the major functions of the software and their internal processing requirements. Specify any external files or databases to which design software must have access. Include design limitations or bounds specified in the System Specifications. Define the style and standards of the system and user documentation, and provide an outline for a user's manual.

2.0 Documentation

List relevant documents, reference material, specifications and manuals that apply to, or are referenced by, the Preliminary Design.

3.0 Design Overview

Provide a top-level, or global, definition of the software structure, data flow, and logical hierarchy. Include a discussion of the control and processing relationships among the major functions of the software.

Two types of diagrams should be included. The first should show the structure of the software. The diagram should be accompanied by a detailed discussion of the logical and hierarchical organization and the structure of the data files (for example, any tables, stacks, queues or lists). See *Figure 12: Hierarchical Program Structure.*

The Preliminary Design describes the overall structure of the system, including the flow of the data and control of that flow through the structure.

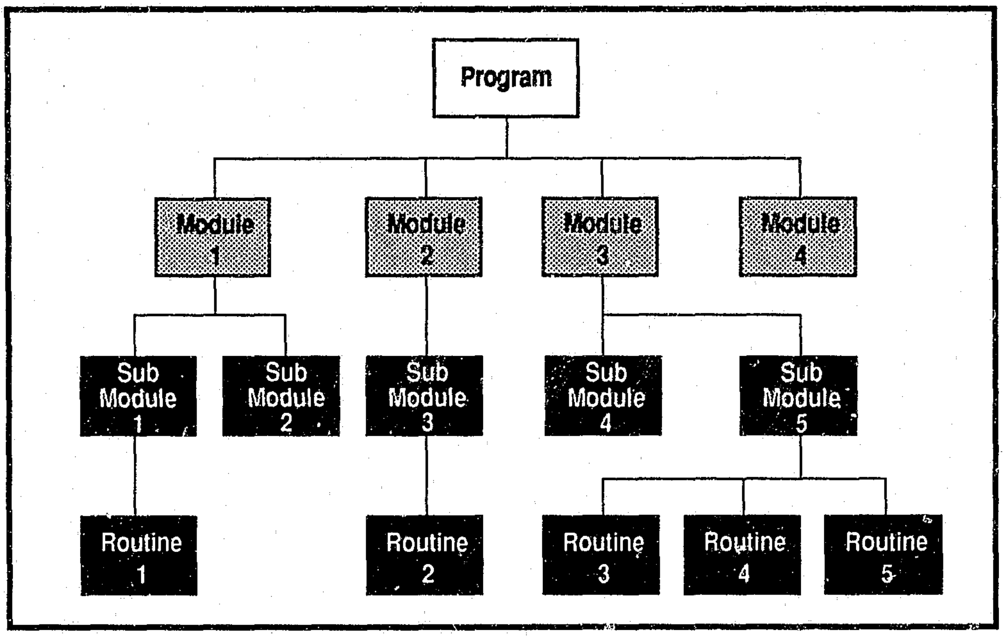


Figure 12: Hierarchical Program Structure

The second type of diagram depicts critical data flow paths. (See *Figure 13: Sample Data Flow Diagram.*) The diagram should be accompanied by a detailed discussion of the entry, transformation and exit of the data as it flows through the system. Refer to the hierarchical program structure diagram (Figure 12) as appropriate.

4.0 Description of Modules

Provide a detailed description of each module in the system. (Some parts may have to be completed during the Detailed Design.) When complete, these procedural descriptions of the modules can be translated into code during the coding phase.

4.1 Overview

Provide a summary description of the module, represented by a flow chart, Pseudo Code or Structured English, or box diagrams. Include a description of the inputs, outputs and processing for the module. Algorithms or equations should be placed in an appendix. Discuss interfaces between this module and other modules in the system. Identify any other modules called by this module.

4.2 Additional Information

Provide additional commentary to assist the coding personnel, such as information on bounds and limits, error-handing guidelines and performance characteristics.

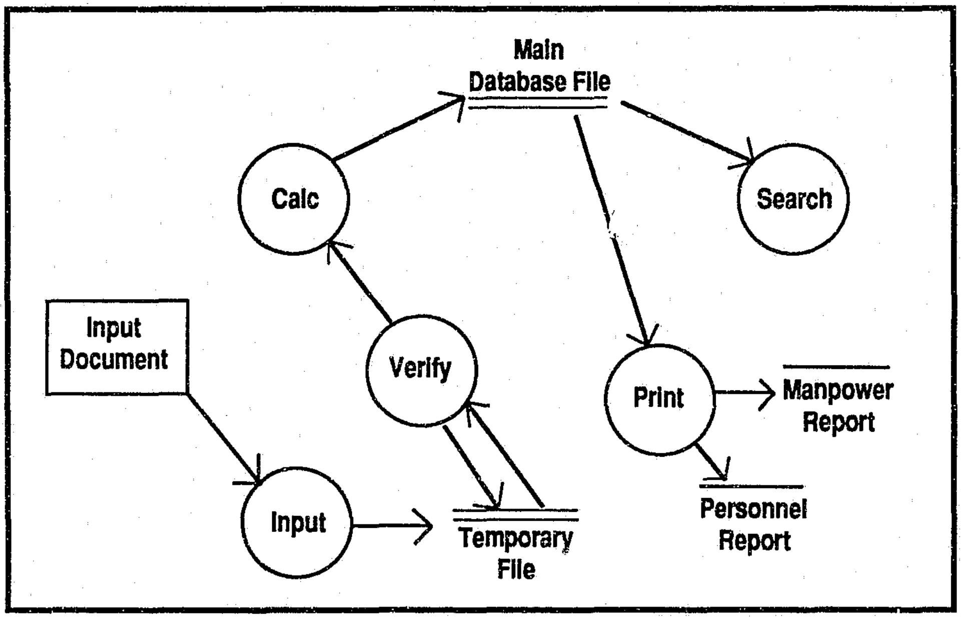


Figure 13: Sample Data Flow Diagram

5.0 File Structure and Global Data

Describe files that are permanent such as a master name file, and those files which exist prior to or after the execution of the software, such as input files and output files. Include the file name, allocated size in bytes, access method, relationship to other files, and whether the file is modifiable or static.

Describe files that are temporary in nature or which exist only during the execution of the software (for example, sort files, print files or queues). Describe global data elements and global memory variables. It is helpful to use a naming convention when defining global and local data files and memory variables. For example, memory variables might be expressed in capital letters with names starting with "M" ("M_NAME" or "M_ADDRESS"). Local data files might be defined in lower-case letters (for example, "m_name" or "m_address").

Much of this information may be provided by supplying two cross reference matrices. The first matrix references the names of the files with the names of the modules, showing which files are accessed by each module. The second matrix depicts data or memory variables accessed by each module (both locally and globally).

6.0 Compliance with End User Requirements

Provide a cross reference matrix of the module names and the corresponding requirements as stated in the End User Requirements document. The intent is to assure compliance of the module function with the end user's requirements.

7.0 Testing Guidelines

Provide general guidelines for Unit Testing, the overall strategy for software integration, and special capabilities required for validation testing. Detailed procedures will be provided in the Test Specifications Document, which is developed in the Detailed Design stage.

For each module, present an overview of the Unit Testing procedures, which will be used to verify compliance with the bounds specified for the module and to verify the proper functioning of the processing within the module. Identify the types of error testing to be applied to data entry. Include a reference to any specialized software required.

8.0 Integration Strategy

Discuss the integration strategy (either Top-down, Bottom-up, or combined) for integrating the individual modules into the completed system. Identify clusters of modules to be integrated as a functional group.

9.0 Special Tools

Identify special tools, software or other capabilities required for the testing.

10.0 Media and Installation Instructions

Identify the type and format of storage media upon which the software will be distributed to the end user. Describe the installation instructions required for transferring the software from the media to the end user's system.

11.0 Appendices

The Preliminary Design appendix should include a Preliminary User's Manual which is usually bound separately. It functions as a guide for the format, tone and required content of the final User's Manual. It also provides a graphic depiction of the required user interfaces for the design and coding phases of the project. Documentation requirements often change when the actual physical document is seen by the end user. Allowing the end user the opportunity to see and make changes to the Preliminary User's Manual reduces the amount of time lost to modifications of the system and documentation in the later phases of the project.

The appendices also should contain any supplemental information, such as tables or code listings.

The Preliminary Design Walkthrough is an exercise used to clarify critical paths, functions and relationships among modules, and interface compatibility with other systems.

Preliminary Design Walkthrough

The Preliminary Design Walkthrough evaluates the feasibility and practicality of the proposed design approach prior to proceeding to the Detailed Design. The Walkthrough traces critical paths through the design, and the members of the review team are responsible for tracking specific operations that the module should perform. If successful, the Walkthrough will clarify critical paths, functions and relationships among modules, and interface compatibility with other systems. It should also identify problems and omissions. The primary objective of the Preliminary Design Walkthrough is to ensure that the system design accurately and comprehensively meets the end user's requirements. All elements of the design model must be traceable to the End User Requirements developed in the Planning Phase.

At a minimum, the Preliminary Design Walkthrough should evaluate:

- System interfaces;
- Software structure;
- Software environment;
- Module functionality;
- Module interaction and control;
- Data flow;
- Data structure;
- Data access; and
- Conformity of the system to the end user requirements.

The Preliminary Design Walkthrough is conducted by the design team and supervised by the Project Manager. It should be attended by representatives of each project team (usually the team leader), the end user and management personnel. Prior to the Walkthrough (preferably a week), all preliminary documentation and the agenda for the Walkthrough should be distributed to the participants for review and comment. The designers should prepare a formal agenda, including the following items:

- System objectives;
- Summary of the System Specifications;
- Alternatives considered but rejected (and why);
- Proposed structure;
- Data flow through the proposed structure;
- Proposed modules;
- Verification of the functional requirements of the proposed modules;
- Bounds and limits on the proposed design; and
- Additional considerations.

Careful adherence to the agenda keeps the Preliminary Design Walkthrough orderly and efficient. A walkthrough is not a complaint session. Accordingly, the tone should remain positive, constructive and directed toward the software design. Minutes of the meeting should reflect all important comments and action items. If properly conducted, the Preliminary Design Walkthrough will ensure that the design has technical merit and that management should approve further development.

If major problems are encountered with the design, the design team must at a later, separate meeting decide upon the proper corrective action and revise the design. In such event, another Preliminary Design Walkthrough must be scheduled.

The primary objective of the Preliminary Design Walkthrough is to ensure that the system design accurately and comprehensively meets the end user's requirements.

Upon successful completion of the Preliminary Design Walkthrough, the System Prototype is produced. It is a working program that uses the approved design of the structure of the system and contains the "empty shell" of all of the modules, but has "stubs" or "dummies" of the actual functions within the modules. The "stub" does nothing more than announce its execution and acknowledge the receipt of any transferred data from the calling module. This operating, albeit stripped-down, version of the proposed software system is known as a shell (or skeleton) and verifies the logical structure, user interface, overall concept and logical flow of the proposed system. The System Prototype:

- Reinforces the development team's confidence in the selected design;
- Reinforces management's confidence in scheduling commitments;
- Reinforces the end user's confidence in the human interface;
- Demonstrates the functionality of the intermodule interfaces; and
- Provides a test bed for the integration of completed modules.

Detailed Design

The major objective of the Detailed Design Phase is to create a design of the entire program which precisely defines the function and internal procedures of each module in the system. The entire program must be completely described in sufficient detail that the programming task is simply a matter of converting the Detailed Design description into code.

The Detailed Design represents the complete system "on the drawing board." For this reason, the comprehensiveness of the documentation is crucial: what is documented is the design itself. If the documentation is incomplete, there is no design. Before allowing the process to proceed to coding, management must be satisfied that the design process has resulted in a complete operational specification for each computer program and sub-program and that the design has been tested and approved. The Detailed Design Document is the master blueprint for the proposed system, and its completeness and integrity are the keys to the success of the project.

The Detailed Design builds on the structure of the Preliminary Design, refining and elaborating the overall design to establish the design of each individual system module. This is the stage in which the detailed algorithms for each of the modules are written. During this phase (which precedes the creation of programming code) each individual module's algorithms, structure, internal processing, data flow, intermodule interfaces and test specifications will be defined. The problem defined by the user has now been completely translated into a detailed computer solution.

To preserve the integrity of the project documentation, it is important that the internal design of each module be represented using a single, standard technique acceptable to all members of the design team. This internal definition of the modules is accomplished by one of a variety of accepted methods, such as Flowcharts, Pseudo Code, Structured English or Decision Tables.

Each module in the Detailed Design may be defined simultaneously by one or more of several analysts working independently. In fact, it is possible that a given module may even advance through Coding and Unit Testing while other modules are still in the Detailed Design stage. The steps of Detailed Design, Detailed Design Walkthrough, Coding, Coding Walkthrough and Unit Testing may take place at different rates for each module in the overall system. This parallel activity can allow for the integration of the modules in an ordered sequence, if emphasis is placed on completing groups of modules which must be integrated together.

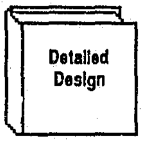
During the Detailed Design Phase, each individual module's alogrithms, structure, internal processing, data flow, intermodule interfaces and test specifications are defined.

Thoroughly reviewing the documented design and testing the design by analytical methodologies are critical to ensuring that the Detailed Design is workable. These activities are executed during the Detailed Design Walkthrough. Accordingly, during the Detailed Design Phase a detailed test plan (to become part of the Test Specifications Document) must be developed.

The Test Specifications Document provides an overview of the testing to be conducted for the duration of the development project. It provides the schedules of tests, the procedures to be used, and the content and format of the test data. During the Design Phase, it is important to plan the order of integration of the modules, the type of integration, the schedule of hardware availability, and the schedule of completion dates. Once these factors have been evaluated and responded to satisfactorily, a resource schedule of all the modules can be created. The schedule contains, along with the module name, a reference to the scheduled Unit Test date, and references to the required test data. The list assures that resources required for the integration testing, such as hardware and software "drivers" are available or are created prior to the required test date.

The distinctions between stages and phases can blur slightly as the individual modules advance through the design and testing phases. Unlike the Project Plan Document or Preliminary Design Document which must be completed as a unit before the project can advance, each module of the design may advance at its own pace until it becomes a part of the integrated system. Different modules of the system may be in varying stages of completion at any given moment due to the complexities of designing, testing, and integrating each module. When the design is tested and approved, it passes to the Coding and Testing Phase.

The Testing Team manager is responsible for maintaining and updating the Test Specifications Document. During the Detailed Design process, the Source Control Library must be monitored and continuously updated by the Source Control Librarian.



Detailed Design Document Format

This document updates and supersedes the Preliminary Design Document.

1.0 System Overview

Generally requires only minimal updating from the Preliminary Design specifications.

2.0 Relevant Documentation

Update to reflect any additional documents, reference material, specifications and manuals added since the Preliminary Design.

3.0 Design Overview

The design overview should not have changed dramatically from the Preliminary Design. It should contain, however, details or clarifications added during the Detailed Design, including modifications to the structure, data flow, and hierarchy diagrams.

4.0 Description of Modules

Update the module descriptions, providing sufficient detail so that coding can commence when the Detailed Design Document is completed.

4.1 Overview

A complete narrative of the procedures within each module updates the Preliminary Design description. The narratives contain information on the inputs, outputs and processing for each module in the system. This is usually represented as a series of charts or matrices. In an appendix, place a detailed Structured English or Pseudo Code listing for each module.

The Detailed Design additions should also include the definitions of the labels and variables in each module, as well as the global data which are accessed by each module. This information is also represented as a series of tables or matrices and included in an appendix. Design notes or comments on the procedures utilized in the design of each specific module also should be included in an appendix.

4.2 Intermodule Interfacing

Provide a narrative description of all of the intermodule interfaces. Include a diagram and/or tabular description of the input and output performed by each module, including variables passed, file I/O, and printing.

Include a diagram of the video displays that will be presented to the end user for input and output. Place hard-copy output or reports generated by the modules in an appendix.

Present a table or matrix of the control or data variables passed to or from any subprograms within each module. Create a single cross reference matrix which specifies all other modules called by each module.

4.3 Data Dictionary

Normally added during the Detailed Design, the Data Dictionary contains descriptions of all global items such as data, variables or buffers; items local to a module, such as data, variables or buffers; and the structure and organization of all files accessed by the modules.

4.4 Additional Information

Provide additional comments which might assist the coding personnel, such as bounds and limits, error-handing guidelines or performance characteristics.

5.0 File Structure and Global Data

This section should not have changed markedly from the Preliminary Design, and should normally require only updating to reflect additions mandated by the the detailed design of the modules.

6.0 Requirements Cross Reference

Completed during the Preliminary Design.

7.0 Testing Guidelines

Completed during the Preliminary Design, it should have changed very little. Include updates.

8.0 Integration Strategy

Completed during the Preliminary Design, it should have changed very little. Include updates.

9.0 Special Tools

Completed during the Preliminary Design, it should have changed very little. Include updates.

10.0 Media and Installation Instructions

Completed during the Preliminary Design, it should have changed very little. Include updates.

11.0 Appendices

The Preliminary User's Manual created in the Preliminary Design should be updated to reflect additional information. The appendices should now contain all of the Pseudo Code descriptions, tables, matiices, diagrams, and other supplemental information required to complete the Detailed Design.

Test Specifications Document Format

1.0 Overview of Testing

Provide an overview of the testing to be conducted. Describe the goal of the testing, types of testing to be performed, overall testing procedures and expected results. Include testing schedules and identify system elements which will be necessary for the test (for example, specialized hardware components).

2.0 Reference Documentation

List all the documents pertinent to the testing specifications.

3.0 Test Plans

Provide a detailed narrative of the testing approach, the schedule, and required software for each phase of testing.

The distinctions between stages and phases can blur slightly as the individual modules advance through the design and testing phase.

3.1 Test Phases

Describe the Unit Testing phase of the coding task. Include a discussion of the test procedures for each module during the coding and unit testing and any schedule limitations or other requirements which should be brought to the attention of the coding team.

Describe the Integration Test phase. Include a discussion of the test and integration procedures for each module, the interdependency of any test upon another test or phase, and any schedule limitations or other requirements.

Provide a summary of the Validation Test phase. Describe the test procedures that will be used for the system and any schedule or personnel limitations.

3.2 Test Schedules

Provide a timeline for the software testing. Include a chart (for example, a Gantt chart) showing the scheduled date of the test, the order of the tests, the interdependency of each testing phase, and the availability of resources required for the testing.

3.2.a Unit Testing Schedule

Provide a chart showing the schedules for the unit testing. Unit testing for specific modules is directly related to the coding of each module, and as such must be constantly revised to maintain timelines.

3.2.b Integration Testing Schedule

Provide a chart showing the schedule for the integration testing. The integration testing for specific modules should be scheduled to coincide with the completion of the Unit Testing of the code, making allowances for the availability of personnel and hardware.

3.2.c Validation Testing Schedule

Provide a chart showing the schedule and order for each of the validation tests.

4.0 Unit Testing

Unit Testing compares individual modules to the design specifications. Using input data, the processing and error-handling paths through the module are tested for unpredictable results. This process results in a completely tested module. This section discusses the tests applied to a module to ensure that it complies to the Detailed Design Specifications. This section may be repeated as often as necessary to describe the Unit Testing of each module.

4.1 Satisfaction of System Specifications

For the module being tested, provide a reference to the System Specifications or Detailed Design Specifications.

4.2 Testing Procedures

Discuss procedures and techniques used in the testing of module functions, the module interface, file access technique(s), and bounds. Identify specialized variables or flags included in the code for testing purposes.

4.3 Testing Tools and Software

Describe special tools, software, resources, or specialized methods used to assure accuracy. Discuss any testing software or drivers created especially for testing purposes. List the sources of testing software in an appendix.

4.4 Test Data

Discuss the format and content of the test data to be used in the testing. Include data sets that will provide for specific test cases, such as range or bound checking, and input validation checking. Test data may be placed in an appendix.

4.5 Expected Results

Discuss the anticipated processing and output results of testing.

5.0 Integration Testing

Integration Testing procedures use the unit-tested individual modules and "builds" to assemble and test the entire software package. This section discusses the testing procedures to be followed when the unit-tested module is integrated into the System Prototype.

Testing is done in three phases: Unit Testing, Integration Testing and Validation Testing.

Integration Testing procedures must be designed to carefully test and discover errors caused by:

- Incorrect coding of the interface as specified in the design;
- Improperly accessing or modifying file and data structures;
- Cascading errors caused by improperly accessing global data;
- Control or module execution sequence errors; and
- Faulty error-handling.

Repeat this section as often as necessary to describe the Integration Testing of each module or group of modules.

5.1 Satisfaction of System Specifications

For the module being tested, provide a reference to the System Specifications or Detailed Design Specifications.

5.2 Testing Procedures

Discuss the procedure and techniques used in the testing. Indicate whether Top-down or Bottom-up Integration is to be used. Specify procedures that will be used to identify and control cascading errors. Describe procedures designed to assure that the entire program structure will be adequately tested during the integration of each module. Specify specialized variables or flags included in the code for testing purposes.

5.3 Testing Tools and Software

Describe special tools, software, resources or specialized methods that will be used to assure accuracy. Discuss testing software or drivers created especially for testing purposes. List the sources of testing software in the appendices.

5.4 Test Data

Discuss the format and content of the test data to be used in the testing. Include data sets that will provide for specific test cases, such as range or bound checking, cascading errors and improper intermodule interfacing. Test data may be placed in an appendix.

5.5 Expected Results

Discuss the expected results of the integration tests and the anticipated output of the tests.

6.0 Validation Test Procedure

Validation testing confirms that the entire integrated software system complies with the system specifications. This section discusses the various tests that are applied to the system.

6.1 Satisfaction of System Specifications

Provide a reference to the System Specifications, Preliminary Design or Detailed Design Specifications that the Validation testing of the system satisfies.

6.2 Testing Procedures

Discuss the procedures and techniques used in the validation tests. Note what each test will demonstrate and how successful execution will validate compliance with the design.

6.3 Testing Tools and Software

Describe special tools, software, resources or specialized methods used to verify the successful execution of the design criteria.

6.4 Test Data

Discuss the format and content of the test data to be used in the Validation Testing. Include data sets that will provide for an evaluation of all aspects of the design criteria.

6.5 Expected Results

Discuss the expected results of the validation tests and the anticipated output of the tests.

6.6 Performance Bounds and Limits

During the Planning and Design Phases, there were performance tolerances and bounds placed upon the system. Specify the allowable range of those tolerances and bounds for the validation tests prior to the executing the Validation Testing.

7.0 Appendices

Contains information grouped separately from the body of the text, for example, the sources of testing software and test data.

Detailed Design Walkthrough

When the design of a module is complete, the designers notify the Design Team Manager that the module is ready for a Detailed Design Walkthrough. A Detailed Design Walkthrough reviews the logic, structure and interfaces of a module prior to coding. The Test Specifications Document is also reviewed and evaluated.

A Detailed Design Walkthrough is an exercise done to review the logic, structure and interfaces of a module prior to coding.

The Walkthrough is usually informal, conducted by another design team member who has been designated a walkthrough reviewer, or by a small walkthrough review team. If the design team is small, the use of a peer committee for the walkthrough is necessary.

The Walkthrough is not intended as a judgment of an analyst's technical skills. The intention, rather, is to ensure a team-oriented consensus on the overall design. All criticism of the system should be constructive.

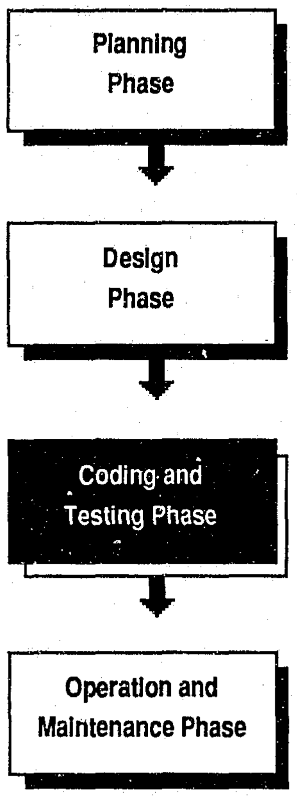
Walkthroughs are usually performed by an experienced analyst familiar with the project and the module's specifications or by a small committee composed of the Design Team and selected representatives from the Coding and Testing Teams. In both cases, the Design Team Manager should act as an impartial moderator and observer of the process. During the Walkthrough, the designer of the module presents the design, provides a step-by-step explanation of the design functions, and answers reviewers' questions. The reviewers examine the documentation of the module and recommend changes to ensure compliance with the Preliminary Design or to correct detected flaws. Typical items examined by the reviewers and discussed by the designer include:

- Correspondence of the module's design to its specifications;
- Practicality of the design in the designated programming language;
- Quality of the design;
- Clarity of structure and adherence to programming conventions;
- Degree of modularity in the design, or structured programming violations;
- Violations of established design conventions; for example, variable name conventions
- Design errors, such as logic flaws;
- Compliance with documentation standards;
- Ease of maintenance; and
- Functionality.

Violations or errors noted by the reviewers are documented and recorded as potential action items by the Design Team Manager. Disagreements between the designer and the reviewers must be arbitrated and decided by the Project Manager or Design Team Manager. When the presentation and review have been completed, the reviewers must document all recommendations.

At this point, a single reviewer, after consulting with the Design Team Manager, either approves the design or suggests revisions. If a walkthrough committee reviewed the design, it votes whether to accept the design or send it back to the Design Team for revisions. In either case, a list of action items is prepared for the designer. Documentation of the walkthrough is placed in the Source Control Library (SCL) by the Design Team Manager.

During the revision process, the Design Team Manager monitors the revisions and updates the design schedule. After the designer completes the required revisions, the Project Manager will schedule another walkthrough. When both the redesign and the walkthrough have been completed and approved, the Design Document is updated. The module can move into the Coding and Unit Testing phase.



Coding and Testing Phase

The Coding and Testing Phase translates the documented, validated Detailed Design into computer program code and validates the code through rigorous testing. The objective of the Coding and Testing Phase is to verify that the coded program meets all requirements delineated in the Planning Process — that is, in the End User Requirements and the System Specifications. In short, the goal is to have the software program perform precisely the work that was desired by the user at the outset of the project. This phase is not complete until there has been a formal approval of the coded software program by the project management. Ultimately, there will be a final acceptance of the program by the end user in the Operation and Maintenance Phase. No documentation is created in this phase.

Coding

The actual process of coding entails converting the logic flows delineated in the Detailed Design Document into executable program code in a specified computer language. It is important to understand that coding of individual modules can be accomplished in parallel by several programmers working independently. Therefore, it is essential that all coding be performed according to the programming standards specified in the Project Plan. Highly recommended are "structured programming" techniques. While a detailed description of the techniques of "structured programming" are beyond the scope of this *Guide*, the reader is encouraged to consult Kernighan and Plauger's *The Elements of Programming Style* for insight into coding techniques and examples of well-constructed control structures.[2]

Coding Walkthrough

The major deliverable for the coding effort is an internally documented, error free, source code listing. The source code listing for each completed module must undergo a walkthrough and be approved before it can advance to Unit Testing.

The Coding Walkthrough is an informal review of the source code by the Coding Team Manager or the programmer with responsibility for the coding effort. Conducted in the same manner as the Detailed Design Walkthrough, the Coding Walkthrough assures compliance of the code with the Detailed Design specifications, including the use of internal documentation and structured programming standards. It is essential that the Detailed Design and the code correspond to each other. If no errors or flaws are found, the coded module is approved and returned to the coder for Unit Testing.

[2]B. Kernighan and P. Planger, *The Elements of Programming Style*, (New York: McGraw-Hill, 1974).

When design flaws are discovered in the walkthrough, it is necessary to return the module design to the Detailed Design phase. A Problem Information Form must be created and a Change Request Form must be forwarded to the Change Control Board as quickly as possible. Documentation of changes is required.

Alpha Testing

Unit Testing, Integration Testing and Validation Testing form what is called Alpha Testing, which confirms that the finished product conforms to the design specifications. Unit Testing focuses on individual modules, while Integration Testing focuses on the modules as a group. Validation Testing focuses on the assembled software to ensure that it meets all the System Specifications prior to end-user testing. Software testing in a structured environment verifies the accuracy of the coding and its conformance to design specifications. It also establishes quality control standards for future maintenance of the software. Responses to standardized inputs are tested to ensure that the software performs reliably. Defective software errors generally falls into three categories: it does not meet specifications, does not match the documentation, or gives unexpected or inconsistent results.

Unit Testing

The coder initiates Unit Testing by specifying the types of tests, the test data and expected results for a given module. Test cases and data are then generated, based upon the requirements in the Test Specifications. Test data should include both "in-bounds" and "out-of-bounds" conditions — that is, data that is both reasonable and unreasonable. Test results should be compared to the expected results.

Unit testing should not begin until the intermodule interfaces have been correctly established and all modules demonstrate correct handling of all conditions. Programs known as "drivers" may need to be constructed by the Coding Team to feed the data to the module in an acceptable format or to record the processing results.

Source code listings generated for both the driver software and the source module should be dated and filed in the Source Control Library (SCL). Source code listings are continually updated during Unit Testing until all tests are successfully completed and a final listing, driver test code, and test results are entered into the SCL. Test cases created during Unit Testing may be used again to form a subset of tests in Integration Testing.

Integration Testing

Integration Testing procedures use the unit-tested individual modules and "builds" to assemble and test the entire software package. Integration is accomplished by replacing the "stubs" in the System Prototype's structure with the unit-tested modules. During Integration Testing, the Test Specifications Document is used to guide the order, schedule and the methodology of testing.

Unit Testing compares individual modules to the design specifications. Test cases are generated and may be used again to form a subset of tests in Integration Testing.

Integration testing may be performed with either a "Top-down" or "Bottom-up" technique, or a combination of the two. In Top-down testing, individual stubs are replaced so that modules highest in the hierarchy of the system are tested first, followed by all subordinate stubs in the functional module. This approach is illustrated in *Figure 14: Top-down Integration.* The Top-down technique allows all of the modules that form a major function to be tested and demonstrated individually. A major data input module might be suitable for Top-down integration.

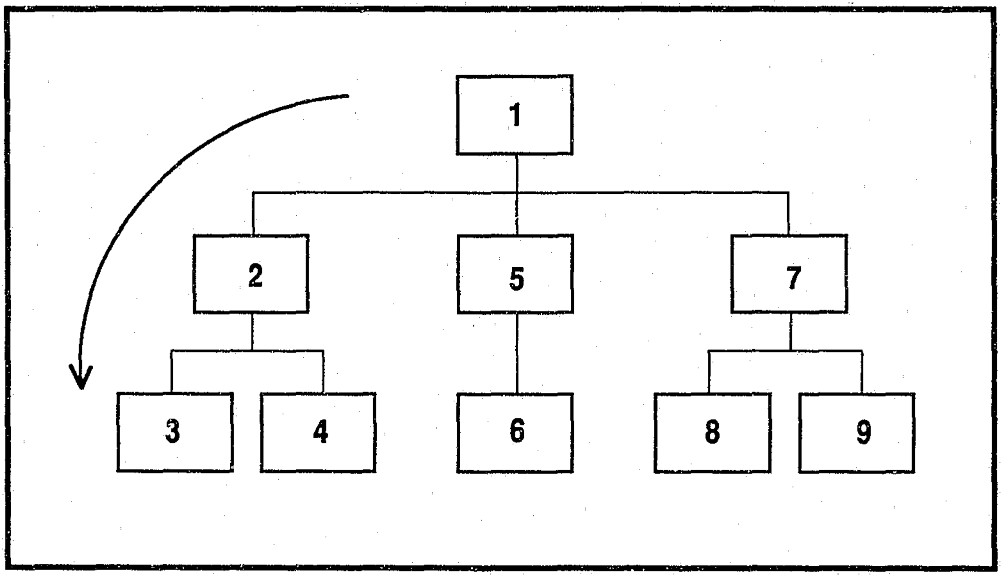


Figure 14: Top-down Integration

The Bottom-up approach integrates one or more modules into a group of modules which is then treated as a single entity or build. The individual modules within a build are tested, and then the builds are tested. Individual modules and builds are then integrated into the overall structure. A build usually replaces a single stub on the system prototype. A posting or update routine might be suitable for Bottom-up integration. (See *Figure 15: Bottom-up Integration.)*

In complex applications or applications with a large number of modules, it is not always possible or desirable to complete the integration of all modules at the same time. An acceptable compromise is to test functional subsets independently and then integrate the subset with the overall structure of the program. The entire package should be tested each time a program stub is replaced by a module.

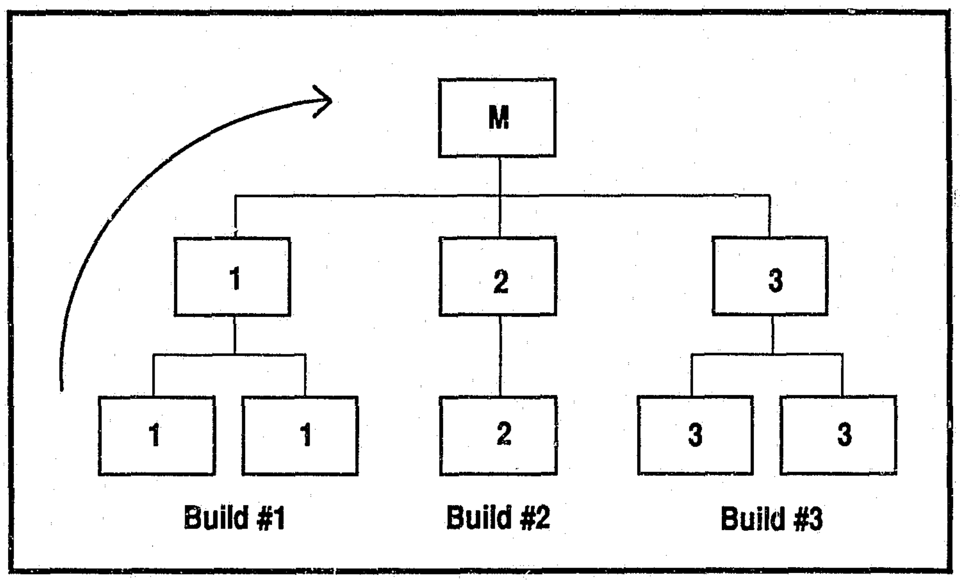


Figure 15: Bottom-up Integration

Scheduling requirements and resources also may make complete integration testing of large numbers of modules impractical, in which case Bottom-up implementation may be required. Usually, scheduling requirements for personnel and equipment or parallel testing requirements force the use of both integration techniques. Moreover, the sequence of integration testing may also be influenced by the function of the modules. For example, all of the modules associated with a major program function may need to be integrated prior to the testing of subordinate modules.

When a new module or build is added to the developing structure, tests pertaining to the new module are conducted, along with all of the old tests for the previously integrated modules. The purpose of this testing is to assure that the addition of the new module has not caused cascading errors.

Integration testing is complete when all modules have been installed, tested and conform to specifications in the Test Plan.

Validation Testing

The purpose of Validation Testing is to test the compliance of the software with the System Specifications by demonstrating the execution of all System Specifications in a complete system.

Ideally, the testing should be conducted by representatives of the end user and technical staff not involved in the design or software coding. Validation test results are then evaluated by the end user and the development team.

A Validation Test Plan, listing features and functions to be validated, is prepared and distributed to the validation testing personnel. The plan is based on the criteria in the System Specifications and relevant sections of the Test Specifications Document. In most cases, the Integration Test plans and procedures form the basis for the Validation Test Plan and, in some cases, may be sufficient to establish the satisfactory Validation Testing.

Validation Testing confirms that the entire integrated software system complies with the Systems Specifications. This testing should be conducted by representatives of the end user and technical staff not involved in the design or software coding.

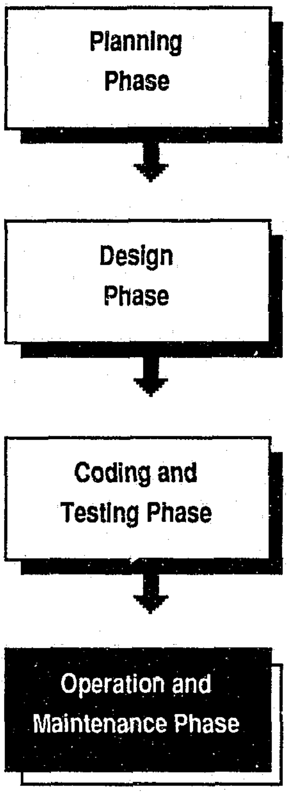
If Validation Testing is successful, the end user should formally sign a document accepting the validated software. Final acceptance and delivery of the completed system software and documentation, however, is usually deferred until after the final phase of testing, known as Site Testing or Beta Testing.

Site Testing (Beta Testing)

Beta Testing is an operational test of the software at selected user sites, which allows the Testing and Design Teams to identify unexpected problems. Beta Testing requires the close cooperation of the end user and the Development Team.

The test plan for the Beta Test must be defined in writing prior to the actual test. This will prevent confusion about what constitutes acceptable measurements of system performance and what procedures will be used to report errors. Typically, a few of the technical members of the end user staff must be shown how to complete PIFs and maintain usage and error logs. A series of meetings should be held at the close of the test to allow the development staff to collect comments about the operation or documentation of the system. At the completion of the Beta Test, the Project Manager decides if the software is ready for general use.

Upon completion of Beta Testing (or Validation Testing if there was no Beta Test), the final update to the Detailed Design is made. The document now represents the Completed Design and becomes the basis for future software maintenance. The Preliminary User's Manual (which was updated during the Detailed Design) is updated to reflect any changes or corrections in the Testing Phase and becomes the Final User's Manual.



Operation and Maintenance Phase

Operational Management of the Software

The final phase of software development, the Operation and Maintenance Phase, begins when the software is installed and is operational at the end user's site. After release of the software to the end user, Operational Management is required. While primarily conducted by the Maintenance Team, Operational Management is the responsibility of both the developers and the end user. Essential management activities include protection of the source code and control of its dissemination, protection of all applications software and operational data from accidental loss, maintenance of software documentation, and communication with the end user. Communication includes information about modifications, the use of alternate peripherals, minor flaws or useful additions and modifications.

Protection of software programs and user data is best accomplished by establishing systematic software management procedures, including regular backups and the creation of an archive of all production programs at an off-site location. The use of serialized and registered master diskettes or tapes, the registration of user sites, and continual revisions in both the software and documentation will discourage unauthorized copying of the software.

Maintenance

Maintenance in the form of required revisions and upgrades requested by end users can be accommodated by careful planning. Moreover, if the software has been developed according to structured guidelines, maintenance can be performed efficiently and economically.

Maintenance can be divided into three distinct activities:

- **Correcting** program errors and faults after the program has been released to the end user;
- **Enhancing** or modifying the features and functions requested by the end user; and
- **Modifying** the program because of a change in the operating environment, such as the installation of a revised operating system or the porting of the software to a new hardware environment.

To perform cost-effective maintenance, the following documentation is essential:

- The System Specifications;
- The Completed Design Document;
- The User's Manual;
- The final version of the internally documented source code; and
- The test data and test results used during development.

These documents should be in the Source Control Library and available on disk or tape. Personnel responsible for the maintenance of the software should review the documentation prior to releasing the software to the end user. This pre-release review is essential because the SCL provides the only source for reference and supporting materials.

Protection of software programs and user data is best accomplished by establishing good software management procedures, including regular backups and the creation of an archive of all production programs at an offsite location.

Procedures for the end user to report software problems and to request changes or enhancements should be established prior to release. Moreover, as in the Development and Testing Phases, the global effect of changes or modifications must be carefully reviewed by the CCB before any action is taken.

Maintenance can be a very expensive part of a software project, often caused by the developer's inability to diagnose all of the potential problems which can develop during operational use of the software. Problems also may develop if the original programming and development personnel left the project during the life of the software.

Performing corrections, enhancements and modifications on-site is very expensive and should be avoided if possible. On-site maintenance must be performed at the end user's convenience and upon the end user's system which normally lacks the tools and utilities of the development system, and may be a very different configuration than the one used by the development team.

Appendix

Recommended Reading List

Requirements Analysis

DeMarco, T. *Structured Analysis and System Specification.* New York: Yourdon Press, 1978.

Fairley, R. *Software Engineering Concepts.* New York: McGraw-Hill, 1985.

Orr, K. *Structured Requirements Specification.* Topeka: Ken Orr and Associates, 1981.

U.S. Department of Justice, Bureau of Justice Statistics. *The Criminal Justice Microcomputer Guide and Software Catalogue*, by SEARCH Group, Inc. Washington D.C.: U.S. Government Printing Office, 1988.

Weinberg, G. *An Introduction to General Systems Thinking.* New York: Wiley-Interscience, 1975.

Weinberg, G. and Weinberg, D. *On the Design of Stable Systems.* New York: Wiley-Interscience, 1979.

Design

Connor, D. *Information System Specification and Design Road Map.* Englewood Cliffs: Prentice-Hall, 1985.

Gane, C. and Sarson,T. *Structured Systems Analysis: Tools and Techniques.* Englewood Cliffs: Prentice-Hall, 1979.

Higgins, D. *Program Design and Construction.* Englewood Cliffs: Prentice-Hall, 1979.

Jackson, M. *System Design.* Englewood Cliffs: Prentice-Hall, 1983.

Martin, J. and McClure, C. *Structured Techniques: The Basis of CASE.* Englewood Cliffs: Prentice-Hall, 1988.

Myers, G. *Composite Structured Design.* New York: Van Nostrand, 1978.

Orr, K. *Structured Systems Development.* New York: Yourdon Press, 1977.

Page-Jones, M. *The Practical Guide to Structures System Design.* New York: Yourdon Press, 1983.

Pressman, R. *Software Engineering: A Practitioner's Approach.* New York: McGraw-Hill, 1982.

Stevens, W. *Using Structured Design: How to Make Programs Simple, Changeable, Flexible, and Reusable.* New York: Wiley-Interscience, 1981.

Yourdon, E. *Techniques of Program Structure and Design.* Englewood Cliffs: Prentice-Hall, 1975.

Yourdon, E. and L. Constantine. *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design.* Englewood Cliffs: Prentice-Hall, 1979.

Yourdon, E. and Constantine, L. *Structured Design.* New York: Yourdon Press, 1978.

Page-Jones, M. *Logical Construction of Programs.* New York: Van Nostrand-Reinhold, 1974.

Costing

Boehm, B. *Software Engineering Economics.* Englewood Cliffs: Prentice-Hall, 1981.

Coding

Dijkstra, E. *A Discipline of Programming.* Englewood Cliffs: Prentice-Hall, 1976.

Kernighan, B. and Plauger, P. *Software Tools.* Reading, Massachusetts: Addison-Wesley, 1976.

Kernighan, B. and Plauger, P. *The Elements of Programming Style.* New York: McGraw-Hill, 1974.

Shooman, M. *Software Engineering.* New York: McGraw-Hill, 1983.

Project Management

Brooks, F.P. *The Mythical Man-Month.* Reading, Massachusetts: Addison-Wesley, 1975.

Gunther, R. *Management Methodology for Software Product Engineering.* Reading, Massachusetts: Addison-Wesley, 1978.

Yourdon, E. *Managing the Structured Techniques.* Second Edition. Englewood Cliffs: Prentice-Hall, 1979.

Yourdon, E. *Structured Walkthroughs.* 2nd ed. New York: Yourdon Press, 1978.

Wiest, J. and Levy, F. *A Management Guide to PERT/CPM.* Second Edition. Englewood Cliffs: Prentice-Hall, 1977.

Systems Analysis

DeMarco, T. *Structured Analysis and System Specification.* Englewood Cliffs: Prentice-Hall, 1979.

Orr, K. *Structured Systems Development.* New York: Yourdon Press, 1977.

Testing

Anderson, R. *Proving Programs Correct.* New York: Wiley Interscience, 1974.

Meyers, G. *The Art of Software Testing*. New York: Wiley Interscience, 1979.

Shooman, M. *Software Engineering Design, Reliability, and Management.* New York: McGraw-Hill, 1983.

Yourdon, E. *Techniques of Program Structure and Design.* Englewood Cliffs: Prentice-Hall, 1975.

Maintenance

Glass, R. and Noiseux, R. *Software Maintenance Guidebook.* Englewood Cliffs: Prentice-Hall, 1981.

Halstead, M. *Elements of Software Science.* New York: Elsevire, 1977.

Lientz, B. and Swanson, E. *Software Maintenance Management: A Study of the Maintenance of Computer Software in 487 Data Processing Organizations*. Reading, Massachusetts:Addison-Wesley, 1980.