

NASA Guidelines for Assuring the Adequacy and Appropriateness of Security Safeguards in Sensitive Applications

F. G. Tompkins

September 1984

MTR-84W179

SPONSOR:
NASA
CONTRACT NO.:
NASW-3425
PROJECT NO.:
1915L
DEPT.:
W-27

The MITRE Corporation
Metric Division
1820 Dolley Madison Boulevard
McLean, Virginia 22120

ABSTRACT

The Office of Management and Budget (OMB) Circular A-71, Transmittal Memorandum No. 1, requires that each agency establish a management control process to assure that appropriate administrative, physical and technical safeguards are incorporated into all new computer applications. In addition to security specifications, the management control process should assure that the safeguards are adequate for the application. This document examines the security activities that should be integral to the system development process and the software quality assurance process to assure that adequate and appropriate controls are incorporated into sensitive applications. Security for software packages is also discussed.

ACKNOWLEDGEMENT

The author wishes to thank R.S. Rice of NASA who provided assistance to MITRE during the writing of this report and the following who provided background information: R. Martian, General Electric Company; F. Mayo, UNINET; A. Sorkowitz, Department of Housing and Urban Development; and G. Mevius, Peer Services. The author would also like to thank MITRE associates W.T. Bisignani, B.A. Christoph, H.R. Keough, and S.F. Levitas for suggestions and review of the final report; N. Cosgrove and D. Violet for graphics and publication assistance; D. Chambers and R. Rosenzweig, for editorial suggestions and clerical support.

TABLE OF CONTENTS

	<u>Page</u>
LIST OF ILLUSTRATIONS	viii
1. INTRODUCTION	1-1
1.1 Background	1-2
1.2 Security Issues in the Software Development Life Cycle	1-2
1.2.1 Issue #1--Sufficiency of Review and Approval of Security Specifications and Systems Tests	1-7
1.2.2 Issue #2--What Activities Are Required To Assure the Quality of Application Systems Security Safeguards?	1-7
1.2.3 Issue #3--How Visible Should Safeguards be in the Application Code and Documentation	1-8
1.2.4 Issue #4--Security Safeguards in Packaged Software	1-9
2. THE SOFTWARE DEVELOPMENT LIFE CYCLE AND SECURITY	2-1
2.1 OMB Circular A-71, Transmittal Memorandum No. 1 Requirements for Applications Software Security	2-1
2.2 The Software Development Life Cycle	2-3
2.2.1 The Initiation Phase	2-3
2.2.2 The Development Phase	2-7
2.2.2.1 The Definition Stage	2-7
2.2.2.2 The Design Stage	2-7
2.2.2.3 The Programming Stage	2-8
2.2.2.4 The Test Stage	2-8
2.2.3 The Operations Phase	2-8
2.2.3.1 The Implementation Stage	2-9
2.2.3.2 The Maintenance Stage	2-9
2.3 Software Development Life Cycle Security Activities	2-9
2.3.1 Security Activities	2-10
2.3.1.1 Determine The Sensitivity of the Data/Application	2-12
2.3.1.2 Determine The Security Objective(s)	2-16
2.3.1.3 Assess the Security Risks	2-17
2.3.1.4 Security Feasibility Study	2-25
2.3.1.5 Define Security Requirements	2-26

TABLE OF CONTENTS
(CONTINUED)

	<u>Page</u>
2.3.1.6 Develop the Security Test Plan	2-35
2.3.1.7 Design the Security Specifications	2-39
2.3.1.8 Develop Security Test Procedures	2-46
2.3.1.9 Write Security Relevant Code	2-57
2.3.1.10 Document Security Safeguards	2-61
2.3.1.11 Conduct Security Test and Evaluation	2-63
2.3.1.12 Write Security Test and Evaluation Report	2-64
2.3.1.13 Prepare the Proposed Certification Statement	2-66
3. SOFTWARE QUALITY ASSURANCE AND SECURITY	3-1
3.1 The Cost of Software Errors	3-2
3.2 Software Quality Assurance	3-6
3.2.1 Software Quality Factors	3-7
3.2.2 Software Quality Factors and the Life Cycle	3-8
3.3 The Software Quality Assurance Process	3-8
3.3.1 Software Quality Assurance Baselines	3-11
3.3.2 Reviews and Audits	3-13
3.4 Software Quality Assurance Life Cycle Security Activities	3-14
3.4.1 Security Safeguard Characteristics (Factors)	3-15
3.4.2 Security Assurance Activities	3-16
3.4.2.1 Security Requirements Review	3-16
3.4.2.2 Security Design Review	3-16
3.4.2.3 Security Specifications Review	3-18
3.4.2.4 Security Test Readiness Review	3-19
3.4.2.5 Security Test and Evaluation Review	3-19
4. SAFEGUARD VISIBILITY	4-1
4.1 The Needs of the Application Owner	4-3
4.2 The Needs of Systems Designers	4-3
4.3 The Needs of Systems Developers	4-4
4.4 The Needs of System Maintainers	4-4
4.5 The Needs of Computer Operators	4-4
4.6 The Needs of Data Users	4-5
4.7 The Needs of Data Providers	4-5

TABLE OF CONTENTS
(CONCLUDED)

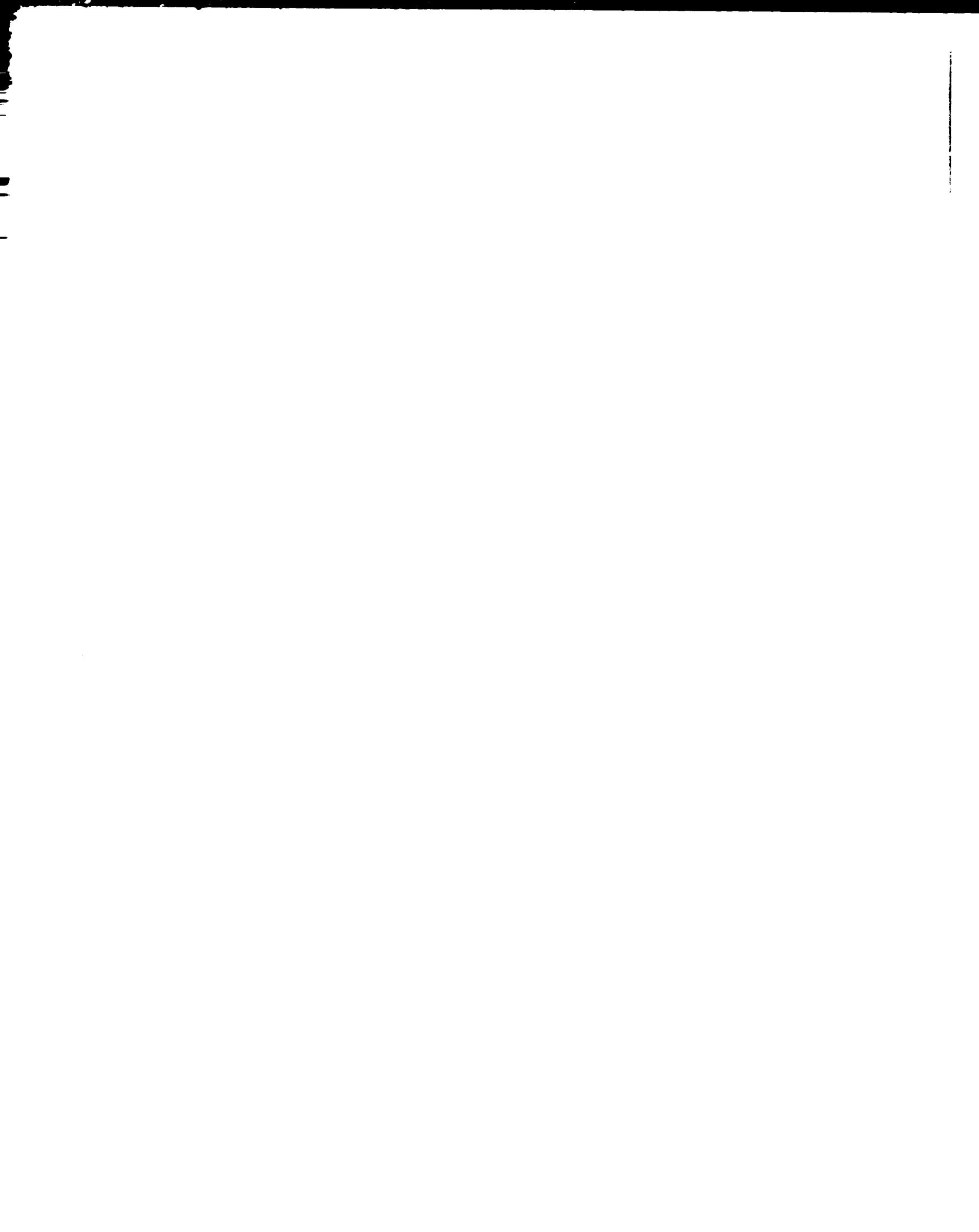
	<u>Page</u>
4.8 The Needs of Data Custodians	4-5
4.9 The Needs of Auditors	4-6
4.10 The Needs for Protection Against Potential Perpetrators	4-6
5. SECURITY SAFEGUARDS IN PACKAGED SOFTWARE	5-1
5.1 System Development Life Cycle Activities for Packaged Software	5-2
5.2 Approaches for Addressing Security in Software Packages	5-6
5.3 Security Assurance and Certification of Packaged Software	5-8
APPENDIX A: SAMPLE SECURITY TEST PLAN	A-1
APPENDIX B: REFERENCES	B-1

LIST OF ILLUSTRATIONS

<u>Figure Number</u>		<u>Page</u>
1-1	ASSURING SECURITY THROUGH THE SOFTWARE DEVELOPMENT LIFE CYCLE, AND SOFTWARE QUALITY ASSURANCE	1-5
2-1	SURVEY OF SOFTWARE LIFE-CYCLE MODELS	2-4
2-2	THE SOFTWARE LIFECYCLE	2-5
2-3	SOFTWARE DEVELOPMENT LIFECYCLE SECURITY AWARENESS	2-11
2-4	SENSITIVE APPLICATION SECURITY OBJECTIVES	2-18
2-5	OUTLINE OF INTRODUCTORY COMMENTS BY MODERATOR AT FIRST THREAT SESSION	2-23
2-6	CERTIFICATION PROCESS	2-38
2-7	SAMPLE OUTLINE FOR A SECURITY EVALUATION REPORT	2-65
3-1	RELATIONSHIPS BETWEEN ERROR CORRECTIONS AND TIME	3-4
3-2	RELATIONSHIPS BETWEEN ERROR CORRECTIONS AND POTENTIAL LOSS OF EXPLOITED ERRORED SOFTWARE SAFEGUARD	3-5
3-3	RELATIONSHIP OF SOFTWARE QUALITY FACTORS TO THE SOFTWARE LIFE CYCLE	3-9
3-4	RELATIONSHIP OF FACTORS TO FILE-CYCLE PHASES	3-10
3-5	SOFTWARE QUALITY ASSURANCE REVIEWS AND BASELINES	3-12
3-6	SECURITY ASSURANCE REVIEWS AND BASELINES	3-17
3-7	CRITERIA FOR ASSESSING SECURITY EVALUATION REPORTS	3-20

LIST OF ILLUSTRATIONS
(CONCLUDED)

<u>Figure Number</u>		<u>Page</u>
4-1	SECURITY SAFEGUARD VISIBILITY REQUIREMENTS	4-2
5-1	THE SOFTWARE LIFE CYCLE FOR PACKAGED SOFTWARE	5-7



1. INTRODUCTION

The Office of Management and Budget (OMB) Circular A-71, Transmittal Memorandum (TM) No. 1, dated 27 July 1978, requires each agency to develop and implement a computer security program. One of the specific requirements of OMB Circular A-71, TM No. 1 is that each agency must establish a management control process to assure that appropriate administrative, physical and technical safeguards are incorporated into all new computer applications. The objective of the management control process is to assure that, in addition to the security specifications (and/or security safeguards) meeting all applicable Federal policies, regulations and standards, the security provisions must be adequate for the application.

NASA has made significant progress in the development and implementation of an agency-wide computer security program in compliance with OMB Circular A-71, TM No. 1. NASA Management Instruction (NMI) 2410.7, "Assuring the Security and Integrity of NASA Data Processing" has been issued. Guidance to the NASA Centers for the development and implementation of NASA Center Computer Security Programs has been incorporated in NASA Handbook (NHB) 2410.1, "Computer Resources Management." Additionally, NASA Headquarters and the Centers have published guidelines in the areas of certification of existing applications software, computer security training, contingency planning and risk management. One of the remaining areas where guidance is required is assuring that appropriate attention is given to security safeguards in the design, development and operations phases of the software life cycle for both internally developed and purchased application software.

1.1 Background

Computer services must be protected not only from physical threats such as damage and theft but also from internal vulnerabilities such as programming errors and misuse by unauthorized users [1]. Inadequacies in the design and operation of computer applications are a very frequent source of harmful effects associated with computers, and in most cases the effort to improve security should concentrate on the applications systems. Security concerns should be an integral part of the entire planning, development, and operation of a computer application. Much of what needs to be done to improve security is not clearly separable from what is needed to improve the usefulness, reliability, effectiveness, and efficiency of the computer application [2]. When system developers, users and data processing management address the security concerns as part of the software life cycle process, there are a number of issues which should be reviewed.

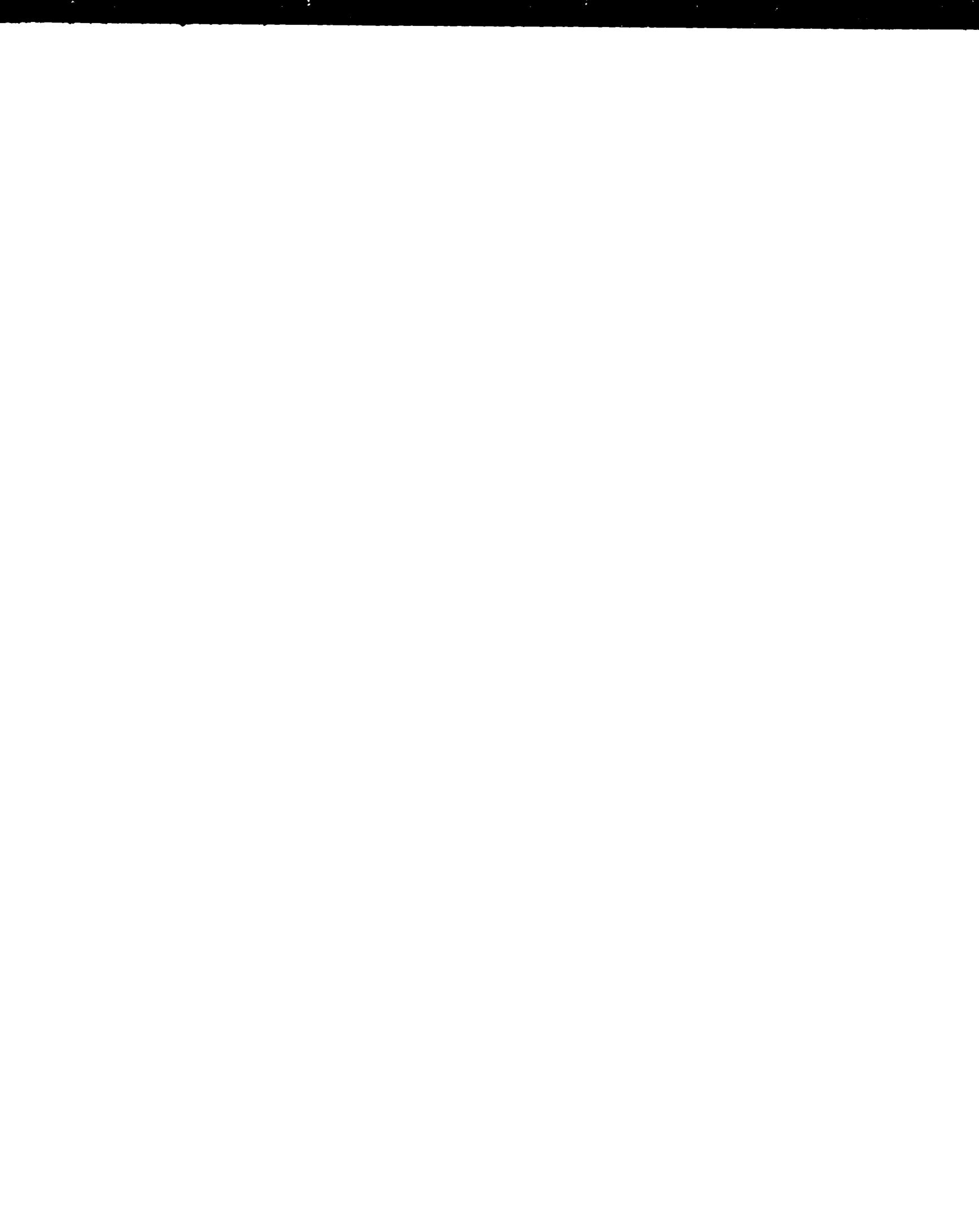
1.2 Security Issues in the Software Development Life Cycle

The software development life cycle (SDLC) is a technique used to divide the system development process into distinct phases with formal management control points placed between and during each phase. The objectives in using an SDLC technique are twofold: to provide a more structured management scheme for controlling costs and schedules, and to ensure proper and responsive communications channels among users, auditors, hardware planning personnel, top management and the data processing personnel responsible for developing the application systems [3]. From a computer security perspective, the SDLC technique, when combined with a software quality assurance process, provides the structure to assure that review points

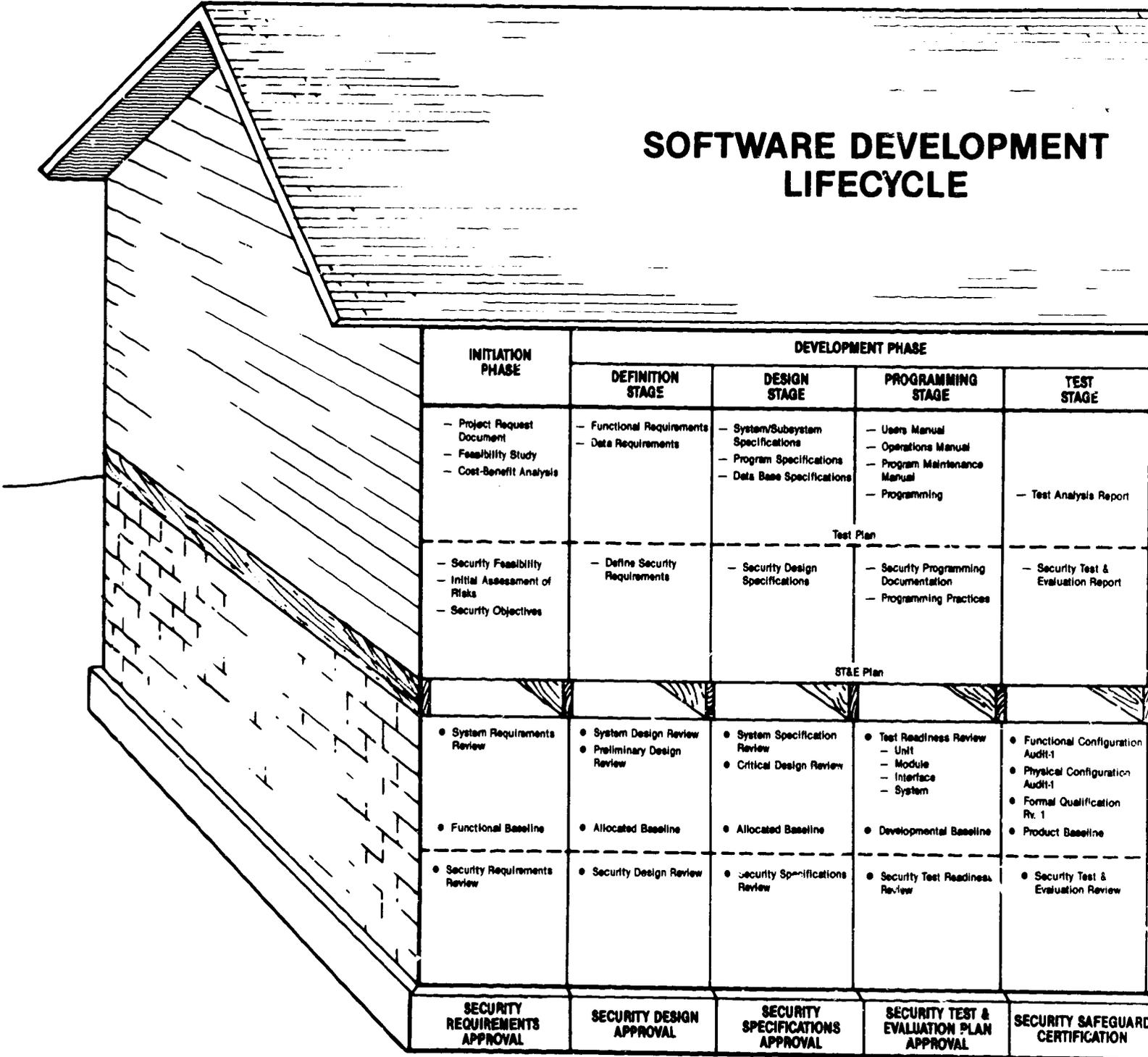
are established to permit computer security management personnel to review and approve the design specifications and the security tests as required by OMB Circular A-71, TM No. 1 (Figure 1-1). While complying with OMB policy is an important consideration, there are a number of other issues that have not been previously well-defined that will ultimately determine whether the security safeguards incorporated into applications systems are operationally effective. The issues that must be addressed during the planning, design, development, testing, integration, implementation and operational stages of the software life cycle are:

- Is review and approval of security specifications and system tests sufficient to ensure that the safeguards are adequate and appropriate? If not, what other reviews and/or approvals are necessary and where in the SDLC should they be accomplished?
- What system development life cycle and/or software quality assurance activities are required to ensure the quality of application system security safeguards?
- How visible should safeguards be in the application code and the documentation? To the user, the developer, the maintainers, auditors and potential perpetrator?

NASA, like most organizations, does not rely solely upon internally or contractor-developed applications. A significant amount of applications software is acquired commercially in packaged form. Adaptations must be made to the system development life cycle to facilitate packaged software. From a security viewpoint, there is concern about how NASA can ensure that packaged software includes the security safeguards that are appropriate and adequate for the applications. This area of packaged software suggests that there is an additional issue that must be addressed:



SOFTWARE DEVELOPMENT LIFECYCLE



DEVELOPMENT CYCLE

DEVELOPMENT PHASE		OPERATIONS PHASE		
PROGRAMMING STAGE	TEST STAGE	IMPLEMENTATION STAGE	MAINTENANCE STAGE	
<ul style="list-style-type: none"> — Users Manual — Operations Manual — Program Maintenance Manual — Programming 	<ul style="list-style-type: none"> — Test Analysis Report 			BASIC SOFTWARE LIFECYCLE
<ul style="list-style-type: none"> — Security Programming Documentation — Programming Practices 	<ul style="list-style-type: none"> — Security Test & Evaluation Report 	<ul style="list-style-type: none"> — Data Control — Employment Practices — Security Training — Security Variance Analysis — Software & Hardware Configuration Control — Contingency Planning 		SECURITY LIFECYCLE
<ul style="list-style-type: none"> — Test Readiness Review <ul style="list-style-type: none"> — Unit — Module — Interface — System — Developmental Baseline 	<ul style="list-style-type: none"> ● Functional Configuration Audit-1 ● Physical Configuration Audit-1 ● Formal Qualification Rv. 1 ● Product Baseline 	<ul style="list-style-type: none"> ● Functional Configuration Audit-2 ● Physical Configuration Audit-2 ● Formal Qualification Rv. 2 ● Operational Baseline 		SOFTWARE QUALITY ASSURANCE ACTIVITIES
<ul style="list-style-type: none"> — Security Test Readiness Review 	<ul style="list-style-type: none"> ● Security Test & Evaluation Review 			SECURITY ASSURANCE ACTIVITIES
SECURITY TEST & EVALUATION PLAN APPROVAL	SECURITY SAFEGUARD CERTIFICATION			SECURITY APPROVALS

**FIGURE 1-1
ASSURING SECURITY
THROUGH THE SOFTWARE
DEVELOPMENT LIFECYCLE, AND
SOFTWARE QUALITY ASSURANCE**

- What activities are required to ensure the inclusion, adequacy and appropriateness of security safeguards in purchased/leased software packages?

1.2.1 Issue #1--Sufficiency of Review and Approval of Security Specifications and Systems Tests

This issue focuses on the need or desirability of security review points beyond or in addition to those required in current OMB policy. Currently, OMB A-71, TM No. 1 requires that security specifications should be approved prior to programming and that system tests be approved prior to using the system operationally. The concern associated with this issue is that the system development life cycle approach includes a number of activities that are accomplished before the generation of specifications that have direct bearing on the ultimate adequacy, appropriateness and effectiveness of security safeguards. The question could also be posed as to whether the review and approval of specification should be accomplished at the preliminary system specifications level or at the detailed (program) specifications level. Some facets of this issue are founded in the variety of terms and SDLC phases used throughout the data processing industry as a whole.

Section 2 of this document discusses the SDLC activities, the activities associated with the SDLC that pertain specifically to the area of security safeguards and the requirements for review of security concerns throughout the SDLC.

1.2.2 Issue #2--What Activities Are Required To Assure the Quality of Application System Security Safeguards?

This issue focuses on the concern that security safeguards, sometimes referred to as internal controls, are most often

judged or evaluated in terms of effectiveness, adequacy or appropriateness. The concern surrounding this issue is that unless the security safeguards that are resident in applications software code are developed with quality as a developmental criteria, they may have flaws that will allow the safeguards to be bypassed or penetrated. Therefore, the cost of loss that may be incurred from exploitation of a flawed safeguard will not only increase the cost to fix the flawed software, but will probably exceed the cost to fix the flaw. The cost to fix a software flaw has been well documented by G.H. Myers and is estimated to be as much as two-and-one-half times more costly to repair in the design, development/test phase and 36 times more costly in the integration phase than if found in the requirements phase of the SDLC [4].

Section 3 of this document will address the areas of how quality software is defined and achieved and how the concept of quality is applied to the case of security safeguards.

1.2.3 Issue #3--How Visible Should Safeguards be in the Application Code and Documentation?

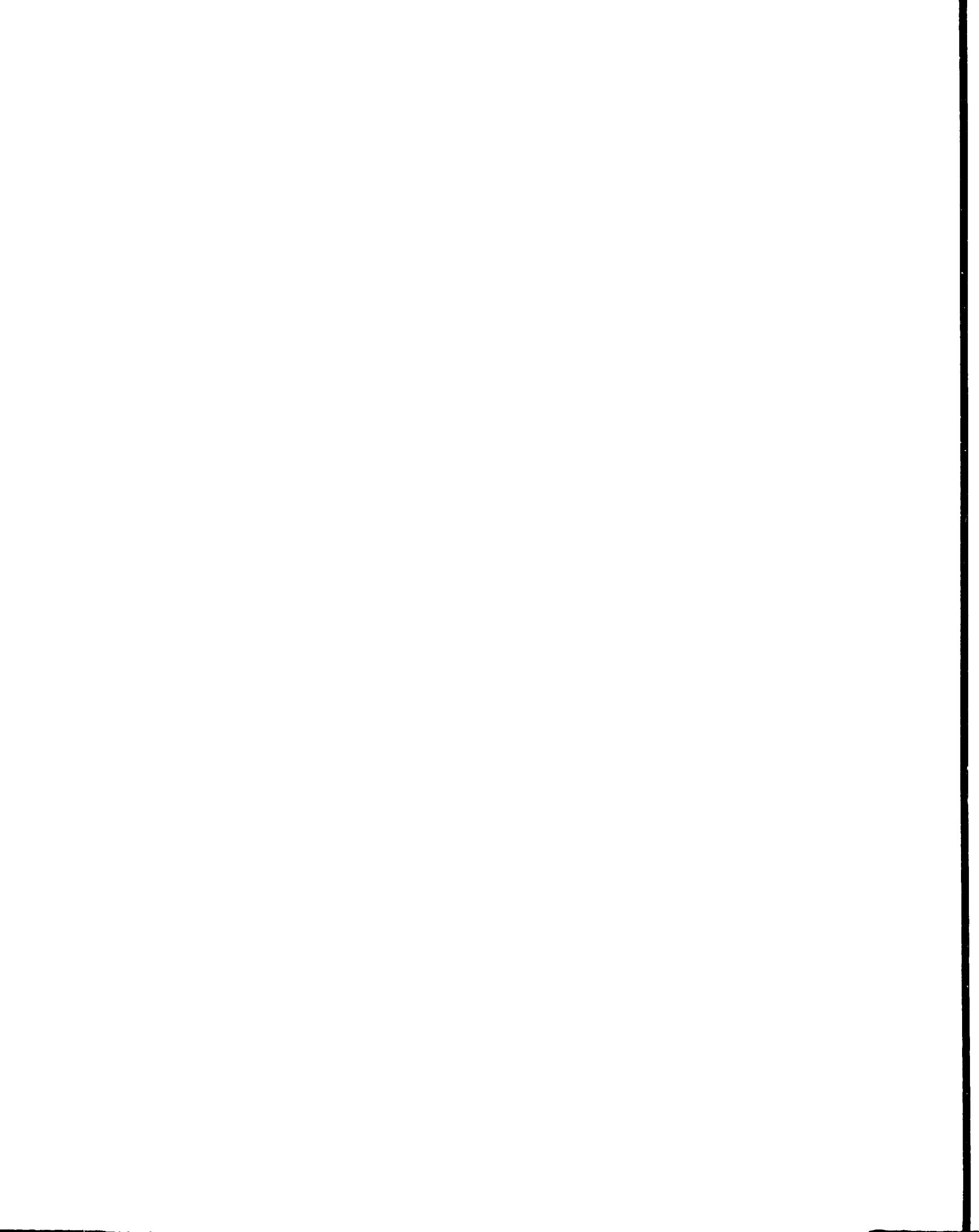
This issue focuses on the requirements by various populations (e.g., users, auditors, programmers, penetrators) to be able to have access to the security safeguards as they appear in the software code, both within the computer and in listings, and in the various pieces of documentation. The concern is that in some cases security safeguards need to be transparent to certain populations so that performance and human engineering attributes are not unnecessarily constrained. At the same time, certain populations require relatively unconstrained or unencumbered access to the safeguards to ensure that the safeguards can be tested, reviewed, maintained, and audited.

Section 4 of this document provides a discussion of the requirements of the various contending populations and the alternatives available for providing the level of visibility that will meet most of the requirements of the population.

1.2.4 Issue #4--Security Safeguards in Packaged Software

This issue focuses on the concern that purchased/leased software packages may not provide, or have sufficient flexibility to provide, security safeguards to meet the security requirements of NASA applications. There is also a concern that insufficient emphasis will be placed on the planning, design, testing, and implementation of security safeguards when acquisition of a software package is chosen in lieu of in-house development.

Section 5 of this document provides a discussion of the modifications that should be made to classic system development life cycle and security activities when addressing the area of security safeguards in software packages.



2. THE SOFTWARE DEVELOPMENT LIFE CYCLE AND SECURITY

While it is common practice for systems developers to think of system functionality first and to delay security concerns until later, many opportunities to include effective controls are lost if not considered early [2]. To assure that system developers consider security throughout the software development life cycle (SDLC), OMB A-71, TM No. 1 requires the establishment of a management control process to assure that appropriate and adequate controls are incorporated into all applications.

2.1 OMB Circular A-71, Transmittal Memorandum No. 1 Requirements for Applications Software Security

OMB Circular A-71, TM No. 1 states that the head of each executive branch department and agency is responsible for assuring an adequate level of security for all agency data whether processed in-house or commercially. This includes responsibility for the establishment of physical, administrative, and technical safeguards required to adequately protect personal, proprietary or other sensitive data not subject to national security regulations, as well as national security data. It also includes responsibility for assuring that automated processes operate effectively and accurately. ...In consideration of problems which have been identified in relation to existing practices, each agency's computer security program shall at a minimum: ...Establish a management control process to assure that appropriate administrative, physical and technical safeguards are incorporated into all new computer applications and significant modifications to existing computer applications. This control process should evaluate the sensitivity of each application. For sensitive applications, particularly those which process sensitive data or which

have a high potential for loss, such as automated decisionmaking systems, specific controls should, at a minimum, include responsibilities for: (1) Defining and approving security specifications prior to programming the applications or changes. The views and recommendations of the computer user organization, the computer installation and the individual responsible for security of the computer installation shall be sought and considered prior to the approval of the security specifications for the application. (2) Conducting and approving design reviews and application systems tests prior to using the systems operationally. The objective of the design reviews should be to ascertain that the proposed design meets the approved security specifications. The objective of the system tests should be to verify that the planned administrative, physical and technical security requirements are operationally adequate prior to use of the system. The results of the design review and system test shall be fully documented and maintained as part of the official records of the agency. Upon completion of the system test, an official of the agency shall certify that the system meets the documented and approved security specifications, meets all applicable Federal policies, regulations and standards, and that the results of the test demonstrate that the security provisions are adequate for the application.

While the terminology used within OMB Circular A-71, TM No. 1 is not consistent with respect to the terms requirements, design and specifications, it is clear that the intent of the overall requirement for a management control process for the security of computer applications software is directed at ensuring that steps are taken to include security concerns and safeguards as an integral, albeit identifiable, part of the software development life cycle process.

2.2 The Software Development Life Cycle

As noted previously, the software development life cycle (SDLC) is a technique used to divide the system development process into distinct phases. Figure 2-1 shows some 15 different software life cycle models. None of these models uses exactly the same terminology for all phases in the cycle. However it is important to note ...the management structure represented by the models is a proven method for enabling a project manager to: (1) estimate the cost/time to complete a system or software project; (2) make use of existing industry and government standards and guidelines; (3) assess the progress of a project at discrete points in the life cycle by conducting formal reviews and audits; and, (4) control system development by requiring go/no-go decision points throughout the life cycle [5].

Figure 2-2 presents a generic life cycle which is based upon the model presented in the National Bureau of Standards FIPS PUBs 38 and 64 [6,7]. The basic software life cycle, as depicted, identifies three major phases: initiation, development and operation. The development phase is divided into four stages: definition, design, programming and test. For the purpose of this document, the operations phase has been divided into two stages: implementation and maintenance.

2.2.1 The Initiation Phase

During the Initiation Phase, the objectives and general definitions of the requirements are established. First, there is an initial user definition activity. During this activity there is a determination of what's currently being done; what needs to be done; understanding the problem; defining of the

ORGANIZATION	PHASES*	INITIATION	DEFINITION	DESIGN	PROGRAMMING	TEST	OPERATION
FLPS PUB 38		INITIATION	DEFINITION	DESIGN	PROGRAMMING	TEST	OPERATION
DOD 7935.1-5		INITIATION	DEFINITION	DESIGN	PROGRAMMING	INTEGRATION, TEST, INSTALLATION	MAINTENANCE
DRAPER LABORATORY (Apr. 1980) (R-1396) NASA		INITIATION	DEFINITION	DESIGN	IMPLEMENTATION	SYSTEM VALIDATION AND ACCEPTANCE	OPERATION
DODD 7920.1		MISS. ANAL. PROJ. INIT.	CONCEPT DEVELOPMENT	DEFINITION/DESIGN	SYSTEM DEVELOPMENT	SYSTEM DEVELOPMENT	DEPLOYMENT AND OPERATION
MARSHALL S.F.C (2/1/79) (Syn Anal. and Int. Lab.)		CONCEPTUAL	REQUIREMENTS	DESIGN	CODE & DEMO	VERIFICATION VALIDATION	OPERATIONS AND MAINTENANCE
TRM(DCA) - 7/15/81		CONCEPTUAL	VALIDATION	DESIGN	CODE & UNIT TEST	INTEGRATION	SYSTEM OPERATIONAL
SOC(ESD) 8/77		CONCEPTUAL	VALIDATION	DESIGN	CODE & CHECK OUT	QUALIFICATION AND ACCEPTANCE	
CTRC		ADV. DEVEL. SYSTEM CONCEPT	VALIDATION	DETAILED DESIGN	FIRST ARTICLE DEVELOPMENT		PRODUCTION/DEPLOYMENT
MITL-STD 1679 NAVY (Weapon Sys.)		PROGRAM PERFORMANCE REQUIREMENTS	PROGRAM DESIGN REQUIREMENTS	PROGRAMMING	PROGRAMMING	TEST	ACCEPTANCE
CSC (DSOM)		REQUIREMENTS	DESIGN	DESIGN & PRODUCTION	CODING	TEST	MAINTENANCE
JPL (S20-15") (6/25/80)		REQUIREMENTS	DESIGN	DESIGN & PRODUCTION	INTEGRATION TEST & TRANSFER	INTEGRATION TEST & TRANSFER	OPERATIONS, SUSTAINING AND MAINTENANCE
IBM (6/80)		SYSTEM DEFINITION	SOFTWARE DESIGN	SOFTWARE DEVELOPMENT	SOFTWARE DEVELOPMENT	SYSTEM ACC TEST	OPERATIONAL SUPPORT
CONTROL DATA		DEFINITION	DESIGN	PROGRAMMING/TEST	PROGRAMMING/TEST	SYSTEM TEST INTRODUC	MAINTENANCE
NATIONAL CASH REGISTER		PROPOSAL/FORMULATION	DESIGN	DESIGN	CODING	UNIT INTEGRATION TEST	ACCEPTANCE DELIVERY MAINTENANCE
NAVAL ELECTRONIC SYSTEMS COMMAND		INITIATION	VALIDATION	FULL SCALE DEVELOPMENT	FULL SCALE DEVELOPMENT	DEPLOYMENT/MAINTENANCE	DEPLOYMENT/MAINTENANCE

NOTE: *Not To Scale

FIGURE 2-1
SURVEY OF SOFTWARE LIFECYCLE MODELS

Initiation Phase	Development Phase				Operation Phase	
	Definition Stage	Design Stage	Programming Stage	Test Stage	Implementation Stage	Maintenance Stage
Pre-ent Request Document Feasibility Study Cost-Benefit Analysis	Functional Requirements Document Data Requirements Document	System Subsystem Specification Program Specification Data Base Specification	Users Manual Operations Manual Program Maintenance Manual	Test Plan Test Analysis Report		Change Requests

(Source: FIPS PUB 38)

**FIGURE 2-2
THE SOFTWARE LIFECYCLE**

scope, objectives and operating environment; definition of functional, performance, and methodological requirements; and, acceptance criteria.

The second activity conducted during the Initiation Phase is evaluation and initiation of necessary documents to formally commence the software development project. This activity includes performing a comprehensive study of technical, operational and economic feasibility; performance of a cost/benefit analysis; analysis of general design approaches, and generating a development plan.

Documentation produced during the Initiation Phase requires user involvement to define the project and its worth. Typically, a Project Request Document is developed as a means for the user organization to request the development, procurement or modification of software or other ADP-related services. It serves as the initiating document in the software life cycle, and provides a basis for communicating with the requesting organization to further analyze requirements and assess impacts. The second document produced is usually the Feasibility Study Document. The purpose of this document is to provide: (1) an analysis of the objectives, requirements and system concepts; (2) an evaluation of alternative approaches for reasonably achieving the objectives; and (3) identification of a proposed approach. The third document, Cost/Benefit Analysis provides managers, users, designers and auditors with adequate cost and benefit information to analyze and evaluate alternative approaches.

All documentation is widely reviewed and is followed by a management decision of whether to continue on to the

Definition Stage. For external procurements, a Request for Proposal (RFP) is issued, proposals are evaluated and a contract is awarded.

2.2.2 The Development Phase

During the Development Phase, the requirements for software are determined and the software is then defined, specified, programmed and tested. The Development Phase is broken down into four stages: Definition, Design, Programming and Test.

2.2.2.1 The Definition Stage

The activities during the Definition Stage include: translation of the user requirements into detailed function requirements and a functional architecture defining the operating environment, functional modules, inputs, outputs, processing requirements and system performance requirements (as needed to meet user performance requirements); definition of data requirements; completion of a general top-level design; definition of functional interfaces (man/machine, system/system, function/function); identification of required equipment; and, planning for development activities. Documents typically produced during the Definition Stage include the Functional Requirements Document and the Data Requirements Document.

2.2.2.2 The Design Stage

During the Design Stage, the Systems and Program Specifications are developed. Activities at this point include: designing the system to meet functional requirements; dividing functional modules into program modules identifying the inputs, processing

and outputs of each; definition of the control and data structures and protocols; and, specification of interfaces in detail. Documents typically produced during this stage include the System/Subsystem Specifications, Program Specifications, Data Base Specifications and the Test Plan.

2.2.2.3 The Programming Stage

During the Programming Stage, the software is coded and debugged. Activities may include obtaining of the required hardware; writing, testing and debugging of software programs; preparation of manuals; and, completion of test procedures. Documentation typically produced during the Programming Stage includes the Users Manual, Operations Manual, Program Maintenance Manual and the Test Plan.

2.2.2.4 The Test Stage

During the Test Stage the software is tested and the related documentation is reviewed. The software and documentation are evaluated in terms of readiness for implementation. Activities include: performance of integration and acceptance testing; training of users and operators; installation in the operational environment; data base conversion; and testing in the operational environment. Documentation produced during this stage is the Test Analysis Report.

2.2.3 The Operations Phase

During the Operations Phase, software is maintained and enhanced as additional requirements are identified. The Operations Phase can be viewed as two distinct stages: Implementation and Maintenance.

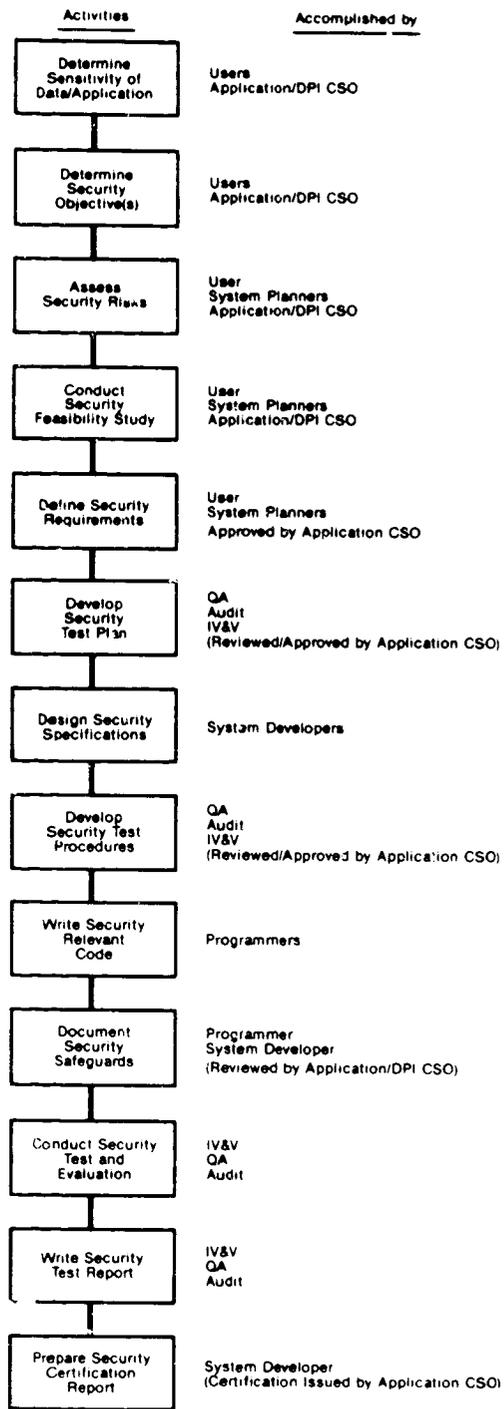
and efficiency of computer applications [2]. While security concerns should be integrated in the life cycle of a computer application, the steps taken to ensure the appropriateness, adequacy and reasonableness of security safeguards should be separately identifiable activities within each stage or phase of the SDLC.

2.3.1 Security Activities

System planners, developers and users should accomplish a series of security-related actions throughout the SDLC. While the order in which the actions should be accomplished is presented sequentially, it should be recognized that there will be much interaction between a particular step and the preceding steps. The process for incorporating security safeguards in an application is not substantially different from the SDLC activities identified in Section 2.2. It should also be noted that if during development, any change occurs in a software requirement or specification, the change must be reviewed to determine if coincidental changes are required in the security requirements or specifications.

The security activities (Figure 2-3) to be completed throughout the SDLC are:

- Determine Sensitivity of Data/Application
- Determine the Security Objective(s)
- Assess the Security Risks
- Conduct a Security Feasibility Study
- Define the Security Requirements
- Develop the Security Test Plan



**FIGURE 2-3
SOFTWARE DEVELOPMENT LIFECYCLE SECURITY ACTIVITIES**

- Design Security Specifications
- Develop the Security Test Procedures
- Write Security Relevant Code
- Document Security Safeguards
- Conduct Security Test & Evaluation
- Write Security Test Analysis Report
- Prepare Security Certification Report

Much of the information provided in the description of the security activities is presented in FIPS PUB 73 [2]. The information is incorporated herein to provide the reader with a complete discussion.

2.3.1.1 Determine the Sensitivity of Data/Application

The degree of sensitivity of an application system depends upon the data it will process and/or the types of functions to be accomplished by the software. For example, data may be personal in nature, represent valuable tangible assets such as high-dollar value inventory or represent real dollars. Application processes which perform critical operations may include formulas or algorithms that must always be executed exactly the same, such as engineering calculations or on-board software for a space vehicle.

FIPS PUB 73, Section 2.3, provides categories of sensitive systems with some examples of the types of applications that would fall under each of the categories. The categories and examples are:

- Applications Providing General Processing Support - The primary concern is for accidents, errors and omissions. Health, safety, welfare and lives may

depend on the correctness of output from these applications. Thus data integrity, including integrity of the software, is critical.

- Engineering calculations used in aircraft design
 - Query systems that support health care decisionmaking
 - Automated wind tunnel control systems
 - Simulation of the dispersion of toxic substances
- Funds Disbursement, Accounting, Asset Management Systems - These systems frequently involve personal or other confidential data. In these applications, deliberate and accidental acts are a major concern. Data integrity is the major objective. Data confidentiality may also be required.
 - Payroll
 - Financial accounting
 - Procurement support
 - Equipment inventory control
- General-Purpose Information Systems - The simultaneous use by different user populations makes data confidentiality critical as well as data integrity. The generality of such systems and their associated security requirements also make it more difficult to design effective security controls.
 - Generalized data management systems
 - Centralized management information systems
- Automated Decisionmaking Systems - Manual review is more difficult, so that errors made by the automated systems are less likely to be detected before they lead to serious harm. The major objective is rigorous data integrity.
 - Fully automated funds disbursement and accounting systems
 - Automated inventory reordering
 - Automated scheduling for maintenance

- Real-Time Control Systems - These systems have all the security concerns of automated decisionmaking systems plus a rigorous requirement for constant availability and very rapid response times. Basic controls plus automated fault detection, backup, and recovery in conjunction with redundant hardware support may be required.
 - Air traffic control
 - NASA mission control
 - Rapid transit system control
 - Automated production control

- Systems Affecting National Security or Well-Being - These systems must be protected against hostile acts by unauthorized persons who have considerable resources. Data integrity, confidentiality, and ADP availability are all required plus techniques for formal development and verification of controls in operating systems as well as application systems.
 - Military command and control
 - Management of multilevel classified information
 - Integrated electronic funds transfer
 - Nuclear material control and accountability

Additional guidance on determining the sensitivity of the data/application can be found in NASA Handbook (NHB) 2410.1 [10]. For the purposes of the NASA Computer Security Program, a sensitive application is defined as the use of a computer system for processing classified, proprietary, dollar sensitive, time sensitive, or Privacy Act data. All other computer system use, such as that for scientific, technical, research, or development activity, may be considered as a nonsensitive application; however, this does not preclude such uses as being designated a sensitive application if this will provide necessary and useful controls. Designation of a test

and mission control application as a sensitive application is within the prerogative of responsible personnel. For test and mission control applications, it is recommended that security measures be provided for as an element of mission or test plans. This will allow any necessary security measures to be tailored to specific test and mission needs in a manner that provides a sound balance between requirements for controls and for operational flexibility.

To assist in determining whether an application may be sensitive, NHB 2410.1 provides the following guidance. First, careful exercise of judgement is required in evaluating the sensitivity of applications. In those instances where it is not clear that an application is sensitive, it is necessary to weigh the intangible costs of potential loss against the cost to protect the application if it is categorized as sensitive. This type of cost-benefit analysis is especially critical in evaluating research, development, test, and mission control applications. The possibility that applications are sensitive only under certain conditions should not be overlooked. For example, it could be useful to categorize unique mission control applications as being sensitive only during a specific period of the mission.

Essentially, the objective of this step is to determine whether or not the application or the data is sensitive. If the data or application is sensitive, the rationale should be documented. When reviewing the data, one should attempt to determine the potential gain to persons from unauthorized access to or use of the data or the application process. This step should be accomplished by the owner or intended user in concert with the application and/or DPI CSO.

If the application is determined to be sensitive, the next step in the process is to determine the security objective or objectives.

2.3.1.2 Determine The Security Objective(s)

There are two types of events that can have unwanted or undesirable effects on sensitive data or applications; advertent (deliberate) or inadvertent (accidental). The advertent or inadvertent events may result in the modification, destruction, or disclosure of data or applications software programs, or the unavailability of computing resources. A useful approach for assuring that appropriate and adequate safeguards are incorporated in sensitive applications is to establish security objectives that, if achieved, will reasonably mitigate advertent or inadvertent events. Generally speaking, five security objectives should satisfy all types of events and effects: data integrity, application integrity, data confidentiality, application confidentiality and ADP availability.

- Data Integrity - The state that exists when computerized data is the same as that in the source documents or has been correctly computed from source data and has not been exposed to accidental or malicious alteration or destruction [2].
- Applications Integrity - The state that exists when the source and object code are the same as originally developed and certified/accredited or, have been modified and tested in accordance with established standards and procedures and recertified/reaccredited, and have not been exposed to accidental or malicious alteration or destruction.
- Data Confidentiality - The state that exists when data is held in confidence and is protected from unauthorized disclosure [2].

- Application Confidentiality - The state that exists when application source and object code and documentation is held in confidence and is protected from unauthorized disclosure.
- ADP Availability - The state that exists when required ADP services can be obtained within an acceptable period of time [2].

Figure 2-4 provides a guide for determining sensitive application security objectives. First, determine the category of sensitive application. Second, refer to Figure 2-4 and determine the possible security objective that may have to be achieved. It should be noted that this is a preliminary determination. The assessment of risks and security feasibility study may surface concerns or limitations that would require a modification of the security objectives.

This step should be accomplished by the owner or intended user in concert with the application and/or DPI CSO.

2.3.1.3 Assess the Security Risks

The types of controls that will ultimately be incorporated into an application system should be determined based upon the potential loss or harm that could be suffered if the data or the application were modified, destroyed or disclosed or is caused to become unavailable due unauthorized or undesirable events. FIPS PUB 102 [8] provides an introductory discussion of the usefulness of risk analysis for assessing the risks to a new application.

The primary purpose of risk analysis is to understand the security problem by identifying security risks, determining their magnitude, and identifying areas where safeguards or controls are needed. It can also

Effect	Type of Event	Category of Sensitive Application						
		General Processing Support	Funds Disb Accounting Asset Mgmt	Gen. Purpose Information Systems	Automated Decisionmaking Systems	Real-Time Control	National Security or Well-Being	
Modification of Data/Appi	Advertent	AI	DI, AI	DI, AI	AI	AI, DI	AI, DI	
	Inadvertent	DI, AI	AI, DI		DI	AI, DI	AI, DI	
Destruction of Data/Appi	Advertent			DI, AI	AI	AI, DI	AI, DI	
	Inadvertent			DI	DI	AI, DI	AI, DI	
Disclosure of Data	Advertent		DC	DC	AC	AC	DC, AC	
	Inadvertent			DC		AC	DC	
Unavailability of Computing Resources	Advertent		AA	AA		AA	AA	
	Inadvertent	AA			AA	AA	AA	

Legend:

- DI = Data Integrity
- DC = Data Confidentiality
- AA = ADP Availability
- AI = Application Integrity
- AC = Application Confidentiality

**FIGURE 2-4
SENSITIVE APPLICATION SECURITY OBJECTIVES**

be used to determine how many resources to budget for security and, with user inputs and policy requirements, can provide the basis for choosing system security requirements.

Risk analysis can also be useful in validating requirements. If requirements are defined to the functional safeguards level, risk analysis can be used to determine whether the protection embodied in the controls reduces expected loss to an acceptable level at acceptance cost.

In the initial assessment of risk, the concern is for the impact and frequency of major failures. The impact of major failures can be described in any convenient terms--dollar value of loss, extent of inconvenience or hardship, lives lost or degree of disruption to the national security or agency mission.

The impact of at least the following failures should be assessed for each major body of information that is to be processed by the proposed system [2].

- Inaccurate Data - Data (programs) could be corrupted with errors, but the system continues to function while producing erroneous outputs. Estimate the potential impact of erroneous actions that might result assuming only that the output of the ADP system is not so obviously out of line with reality that the errors would be noticed. Consider both the impact of a few very serious errors and the cumulative effect of many small errors.
- Falsified Data - An individual could falsify data in order to gain some advantage. The falsifications may be limited only by the fact that they are subtle enough so they are not detected manually. Estimate the total impact that could occur over an extended period of time.
- Disclosed Data - Sensitive data in the system becomes available publicly or to certain individuals. The unauthorized disclosure of data is not necessarily discovered.

- Lost Data or Application Software Code or Documentation - Data, source code, object code or documentation are destroyed or corrupted. Backup versions are nonexistent or not usable, and the data must be reconstructed manually or software code or documentation must be rewritten. Estimate the impact of losing the data, source code, object code or documentation. If manual reconstruction is obviously not feasible and if backup in depth is anticipated, estimate the impact of using old version, inaccuracies, and the temporary unavailability that would result while recovering from an old copy on the assumption that all current or recent backup copies have been destroyed.
- Unavailable Data Services - Estimate the impact if the computer hardware or related equipment in the computer facility (DPI) is disabled and the system is not available until it can be brought up in another facility.

An estimate of the impact of a major security failure is not particularly meaningful without some estimate of how frequently it might occur. Unfortunately, during the initial planning for an application system, it is difficult, if not impossible, to estimate the frequency of a major failure by evaluating the controls in place. However, it is possible to develop rough estimates based upon experience with manual system activities and by looking at the experience of failures or disruptions with similar types of systems. FIPS PUB 73 [2] provides the following guidance:

If the proposed system is generally comparable to other computer applications, then a major security failure of the sort described above can be estimated to occur once in a hundred years. This simple estimate is based on the following line of reasoning:

Available security controls (if they are properly managed) can prevent major security failures from reoccurring as frequently as once every 10 years. On the other hand, any ADP system has several

vulnerabilities against which there is little defense; most systems can be manipulated by any one of several individuals who are in a position of trust--programmers, those responsible for security, the computer operators, and others. There are enough instances of major security failures in computer applications so that an expected frequency of once in a thousand years is very optimistic. The estimate of once in a hundred years is only intended to be accurate within an order of magnitude.

An alternate approach for estimating frequency is to use a low, medium or high frequency rather than orders of magnitude. This approach when used as part of the Threat Team Analysis should provide sufficient data to provide an assessment of the impact of major failures.

A threat team is composed of key employees within an organization who meet as a task force to search for threats and vulnerabilities in a system and who create possible scenarios for attacking the system. Use of such teams is based on the premise that people in the best position to discover how to beat the system are those who work with it every day. The objective is to capitalize on their knowledge. Threat analysis unlocks this potential through the use of a moderator familiar with computer abuse methods. The threat sessions seek a symbiosis between the moderator's general knowledge of typical schemes and the participants' specific knowledge of data processing operations and functional experience with the system under evaluation [1]. The threat team/analysis approach has been used successfully in NASA in the evaluation/certification of existing sensitive systems [12].

Actual threat scenario development is accomplished through a series, usually two, informal team meetings. The first meeting

should be scheduled for two to two and one-half hours. The second meeting should last for one to two hours. At the beginning of the first meeting of the team, a basic set of ground rules and background information must be given. The key elements of this background are summarized in Figure 2-5 [11]. The moderator should give a brief summary of the statistics of computer abuse, an outline of major schemes that have been perpetrated against similar types of applications and how they were accomplished (modus operandi). The objective is to "prime the pump" and stimulate the participants in developing possible attacks against their system.

Flip charts should be used to record the following information:

- What is being attacked or compromised in the system?
- What vulnerabilities would permit the attack to be accomplished?
- What methods or procedures would be utilized?
- What types of safeguards or controls could be used to prevent or reduce loss?
- What is the likelihood that this scenario will work (high, medium, low)?
- What is the impact on the system or organization if the scenario were successfully executed (order of magnitude dollars, delays, etc.)? Note: Quantification of impact while desirable is not mandatory.

One alternate approach is for each threat team participant to keep notes, summarize each scenario and turn the notes over to the moderator for summarizing. The summary would be reviewed by each team member at the second meeting. Another alternative, is for the moderator, or other designated person,

1. Introduction	The moderator is introduced by a NASA management official. Participants have not been informed in advance of the subject of the meeting. The official then leaves. The meeting place should be around a large table in a comfortable room, such as the board room (thus giving status and approval to the project).
2. Summary	The objectives of the study are explained; the time frame and the responsibilities of the participants are outlined.
3. Moderator's Identification	The moderator should establish his position within the group and define his own role, which is that of a resource person. He knows a good deal about computer abuse schemes in general, but little about how this particular organization operates.
4. Why this Organization?	Participants should be put at ease by explaining that the study is simply a precautionary exercise; there is no reason to suspect an on-going perpetration.
5. Why these Participants?	Participants often wonder "why me" at this stage. Again they must be reassured. They have been selected for the study because of their knowledge and experience; they are the people who "really know how this business works."
6. Warning about Secrecy	Participants are asked not to discuss the subject or content of the meetings outside the group.
7. Schedule	The group will meet for several hours. A transcript will be prepared and circulated for changes. A second meeting of the same group is scheduled in about one week, and a final report circulated in the same fashion.

**FIGURE 2-5
OUTLINE OF INTRODUCTORY COMMENTS
BY MODERATOR AT FIRST THREAT SESSION**

to act as a recorder, to keep a "transcript" of the meeting. The transcript would then be reviewed by each participant after each meeting and finalized as a consensus record.

The second meeting should be conducted within one to two weeks after the initial session. The initial session provides a "sensitizing" of the participants. Between the two sessions, participants can review the proposed system with some new perspectives and will usually provide additional scenarios. At the conclusion of the second meeting, conclusions about the scenarios should be agreed upon by all participants. A report of the sessions should be written. The report should be closely held, distribution extremely limited and copies protected.

Upon completion of the threat scenario analysis exercise, it is suggested that the security objectives be reviewed to determine if the preliminary security objectives are still valid. When reviewing the security objectives, the following items should be considered to ensure that major concerns about security have been addressed [2].

- Source data accuracy - Will the data supplied to the ADP system be accurate and complete enough to support its intended uses without harmful side effects?
- User identity verification - Can users of the systems be adequately identified and authenticated so they can be held accountable for their actions?
- Restricted interfaces - Are the user interfaces to the system sufficiently restricted so that adequate security is feasible?
- Separation of duties - Do the boundaries between ADP and related manual activities provide maximum separation of duties and independent review?

- Facility security - Is the proposed processing facility adequately secure?

The next step in the security life cycle is to determine the types of controls that should be incorporated into the application that will achieve the security objectives.

2.3.1.4 Security Feasibility Study

The purpose of the security feasibility study is to determine if controls are available to meet the security objectives, how well they will satisfy the objectives, whether the controls should be preventive, detective or recuperative in nature and what mix of administrative, physical and technical controls is most feasible. Cost, performance degradation, and impact on requirements for user friendliness should be considered. In other words, what types of controls are appropriate for the proposed application.

A key to the feasibility study is the use of an appropriate methodology to analyze the proposed application to determine what security controls are available and how well they meet the security objectives. Brill, in "Building Controls Into Structured Systems" [13], divides application controls into three major classes: controls over inputs, controls over processing and controls over output. Brill's methodology is based on a hierarchical approach that leads the user through a tree structure to address a variety of control issues such as input authorization, internal data movement, operator intervention, and output distribution.

At this point in the life cycle with the determination of sensitivity completed, the identification of security

objectives, the assessment of risks and security feasibility completed, definition of security requirements is the next step.

2.3.1.5 Define Security Requirements

Definition of security requirements takes place during the definition stage of the development phase of the system development life cycle. The term "requirements" can be used at many different levels. The requirements defined at this point should include everything that the users and other responsible parties want to require of the application software. The security requirements should be expressed in a way that permits the software designers to choose the best way of implementing them. It should be remembered that security controls can be enforced either by software or by physical or administrative procedures. For example, data integrity can be checked by human review or by automated bounds and consistency checks. When possible, it is recommended that controls be implemented in software for the following reasons: once controls have been automated, the continuing cost to enforce the controls is usually lower than when enforced manually, and, automated controls will be applied more consistently.

It should be noted that FIPS PUB 73 [2] indicates that the documentation of security requirements constitutes the security specification called for in OMB Circular A-71, Transmittal Memorandum No. 1, paragraph 4.c (1). Also, security specifications may be incorporated into the functional requirements document and the data requirements document as called for in FIPS PUB 38 [6], or it may be an independent document.

The first step in defining security requirements is to conduct an analysis of the "current system" to identify and develop an understanding of the principle functions and to identify sources of input and the flow of data through the system. When the current system is reasonably understood, the user and system requirements should be documented. The sources of input and the flow of data through the system are two of the most important sources of data for defining security requirements. Brill [13] indicates that systems analysts can use four different sources to identify controls that belong in the new system: stated user policy, unstated user policy, the current system and external constraints.

- Stated user policy--The best way to begin to determine the controls needed in a system is to ask the users about the need for controls. Users have a genuine stake in the new system as well as thorough knowledge of their own requirements. Users should be asked how they handle errors in the present system, the kinds of errors or problems they suspect they don't know about (that is, those that are slipping through undetected). Users should also be asked about laws or regulations that affect the way system processes must be done as well as questions about the value and criticality of their data.
- Unstated user policy--Users expect comprehensive controls to be built into their system without ever mentioning controls in discussions with the analysis team. For example, the need to test check digits on account numbers may never be stated because the present manual system may not have facilities to permit it. But users may assume that an automated system tests check digits as a matter of course and that you don't need to be told to build such tests into the system. Systems planners must learn the system--and understand the problem--well enough to challenge the unstated assumptions of the users and turn unstated requirements into stated ones.

- The current system--New systems tend to do many of the same things as the systems they replace. Of course, they may do them differently (via a terminal, for example, rather than through batch-produced reports). So, while the specific controls in the new system may differ from those in the old system, there are going to be overlaps. But, you have to look for them, and have to recognize them as controls and to assess their suitability for transplantation into the new system.
- External constraints--Various laws and regulations can directly affect a system. (This area would include agency policy for security, financial accounting, etc.)

FIPS PUB 73 [2] suggests that the following areas be addressed when developing security requirements: identification and definition of systems interfaces (to include responsibilities associated with each interface and a separation of duties), identification of the sensitive objects to be processed, determination of error tolerances and definition of availability requirements.

- Identification and definition of systems interfaces--Identify each job function which is related to the application system. Consider each job function as an interface to the application. Define the nature of the interaction between each job function and the system. Also identify and define any interfaces to other automated systems. Include all job functions (or other automated systems) that are to be supported by this system--even if the people performing those jobs only receive reports from it. Also include all job functions (or other automated systems) that supply information to the application system or that support its operation. Be sure to include critical job functions such as: source data collection, input preparation, data entry, output dissemination, data base administration, system security planning and control, internal audit,

application program maintenance, archival or backup data storage, computer operations, and system programming.

Define the responsibilities of the individuals who interact with the application system through each defined interface. Identify the constraints on the use of each interface that must be enforced if security is to be preserved. Consider the likelihood of errors occurring in the use of the interface and identify the consequences. Consider the consequences of deliberate misuse of the interface. Identify the management and administrative controls that will be available to ensure that the interface is used properly.

Examine interfaces to ensure that security exposure will be minimized even if an interface is misused. Ideally, any action that could result in serious harm should be checked or approved from an independent interface.

- Identification of sensitive objects to be processed--Identify the objects to be processed--include input data, stored data and output data. Determine the sensitivity and asset value of the data objects. Identify the operations or functions that users will perform on this data.
- Determination of error tolerances--Determine the application's error tolerance by taking into consideration the expected reliability and validity of the data and the intended objectives of the application. For example, funds disbursement or electronic funds transfer systems may have a low tolerance for data error since such errors directly translate into dollars. Real-time control systems such as air traffic control have virtually no margin for error since human lives may be lost. Some management information systems, particularly those used to predict future demands and resource requirements, may not be as susceptible to errors in data. However, algorithms in the code may have less tolerance for errors. The application's tolerance for error and the requirements for maintaining error levels within acceptable tolerance must be defined.

- Definition of availability requirements--Determine the user tolerance for interruption of output data and the potential harm that could be a result due to non-availability of the application output.

The preceding discussion in this section has addressed the collection and analysis of data that precedes the actual documentation of security requirements. Requirements should define what is required by the user not how it is to be accomplished.

FIPS PUB 73 [2] describes some basic controls that can be used to achieve security objectives. It is relevant at this point in this discussion to summarize these basic controls since the descriptions provided, with some modification, can be used as security requirements. The basic controls provided in FIPS PUB 73 are: data validation, user identity verification, authorization, journaling, variance detection, and encryption.

- Data validation--Invalid data may lead to erroneous outputs, can destroy the credibility of the system, demoralize those trying to use it, cause excessive system maintenance costs, and, in extreme cases, cause the system to become unavailable or unusable.

Data validation involves the examination of computerized data to determine if it is accurate, complete, consistent, unambiguous, and reasonable. Direct evaluation methods (discussed below) are not able to find all errors. Data integrity depends on the correctness and integrity of all the activities by which the data is collected and processed. Data validation is a very basic control, but it should only be expected to detect gross errors and it will not compensate for poor control over other aspects of the application system. Data should be validated

during data collection and entry--prior to its use by the system; and, continuously, as new data is generated or used during processing.

Data validation should be required during data entry and during processing. Automated editing and validation should be used in both batch and on-line systems. In batch processing systems, validation routines may run against input data before it is processed. Alternately, validation can occur as each transaction is processed. Transactions that contain errors should be recorded on a file for correction at a later time. On-line systems can provide the data entry personnel with immediate validation information so detected errors can be corrected immediately. During the definition stage the editing and validation technique to be employed is not specified, rather the requirement should state that all data originating from hard copy should be validated prior to the transaction being entered into the system.

For a batch system a typical requirement statement might read:

All source data will be keyed twice and automatically compared with the transcribed source data previously keyed.

For an on-line system where transactions are entered in real-time, a typical security requirement might read:

All keyed transactions (or transactions of a certain type) will be visually verified prior to transmission to the system.

Data may also be validated during processing. Most of the techniques that are appropriate to validation during data entry may be applied during processing. An example of a requirement that related to validation during processing is:

Transaction with errors detected during the data processing phase need to be controlled to ensure they are corrected and reentered in a timely phase.

- User identity verification--Identification occurs when the user provides an identifier--the name by which the user is known to the system. The user's identifier is unique, unlikely to change and need not be kept secret. It is used during processing for authorization controls, variance detection and for other purpose such as accounting and billing. Verification occurs when the individual passes some further test which "proves" that the user is actually the person associated with the identifier. This is also called user authentication.

Examples of requirements for user identification and/or verification might be as follows:

For batch submissions, users must be visually identified by a control clerk and all jobs logged.

For on-line submission of transactions, all users must have an individual identifier and password for initial logging on.

- Authorization--Once a user's identity has been verified, the application may still need to check each request for service to determine whether it is a legitimate request by that user. Some users may be authorized to perform some functions but not others and to have access to perform some functions but not others and to have access to some data but not to other data. In some cases, the authorization decision may depend on not only WHO is requesting what MODE OF ACCESS to which OBJECT, but also on other easily testable conditions. The time of day, the day of the week, previously detected security variances, or other concurrent activity might be used to affect the authorization decision.

An example of this type of authorization when stated as a security requirement might read:

The system must be able to restrict update access to specific time of the day and days of the week.

- Journaling--Journals may be employed to log activities or events external to the operational environment or those internal to the application system. Journals of external events in the operational environment can be maintained manually while journals used to record activities internal to the application must be automated. From a security standpoint, the ideal journal would include a 100 percent recording of all events relating to data, software, and system resources. From a practical standpoint, such a journal may, in some systems, be out of the question since the overhead to record all events would reduce system response to less than acceptable levels of performance. Requirements for journaling should be carefully considered, reduced to formal statements and be stated in positive terms. Items that should be included are:

- definitions of what kind of data is to be protected and how the system will recognize such data,
- the degree of accuracy that is necessary for various types of data, and
- the definition of who is authorized to access protected data and how the system would recognize an authorized user.

Examples of a security requirement for journaling are:

The system will log all accesses by payroll personnel to any employee's payroll record. The system will log all initial log-ons, final log-off at the end of the normal work day and all log-ons and log-offs on weekends.

The system will log all opening and closings of the payroll master file and payroll transaction update files.

- Variance detection--The objective of variance detection is to allow management to detect and react to departures from established rules and procedures that it has determined may constitute hazards. Variance detection acts as a strong deterrent to authorized

users abusing their privileges since they perceive the risk of detection to be unacceptably high. Variance detection is useful whenever it is not practical to prevent the undesirable activity by means of an authorization mechanism. In some cases, there may be no way to determine in advance whether an action should be prevented. The mechanisms required to support variance detection are related to mechanisms needed for other purposes. Recovery, accounting, load-balancing, tuning and the identification of recurring user difficulties all require some of the same capabilities.

A security requirements for variance detection might read:

The system must be capable of providing post-processing analysis of all or selected activity initiated from a given terminal or by a given employee.

The system must provide an interaction capability to identify attempted accesses to restricted files by unauthorized users.

The preceding are examples of static monitoring. Some variances can and should be detected in real-time so that responses can be immediate. An example of a dynamic monitoring requirement might read:

The system must be capable of real-time display at a designated console of the full interactive traffic of any terminal or user.

- Encryption--The applications that are most likely to need encryption are those that transmit highly confidential data across communication lines. Applications that transmit financial transactions or other critical data may also need encryption if some is likely to derive enough benefit from modifying the data during transmission to compensate for the risk and cost of the effort. Encryption of data in storage is an alternative that may be more cost effective than other storage security controls--especially when appropriate support for encryption is readily available.

A security requirement for encryption might read:

All data transmitted between the host computer and remote site will be protected from unauthorized disclosure and modification during transmission.

Following the definition of security requirements, two activities should be initiated: development of the test plan and the design of the security specifications. While they are shown in Figure 2-3 as sequential events, they can be accomplished in parallel since two different groups of people are involved.

2.3.1.6 Develop the Security Test Plan

Testing and evaluation attempts to demonstrate that a system is reliable, meets specifications, and meets the requirements of the user. Sorkowitz [14] provides the following comments on testing:

The main problem in program testing becomes clear when we try to define the word "testing." To many programmers, testing is a process of proving that a program is correct (i.e., the program performs according to specification). However, experience leads us to the belief that there really is no practical way to demonstrate that a program is correct. The best we can say is that at some point in time, there are no known errors. Myers [4] gives a different definition of testing: "Testing is the process of executing a program with the intention of finding errors."

Careful and thorough testing and evaluation can improve system security by uncovering errors, omissions and other flaws in the system's design and code [2]. From a security perspective, the testing of security controls should focus on ensuring that

security controls are invoked when required, that they cannot be easily bypassed, that they are auditable and that they are appropriate in view of the sensitivity of the data or the application.

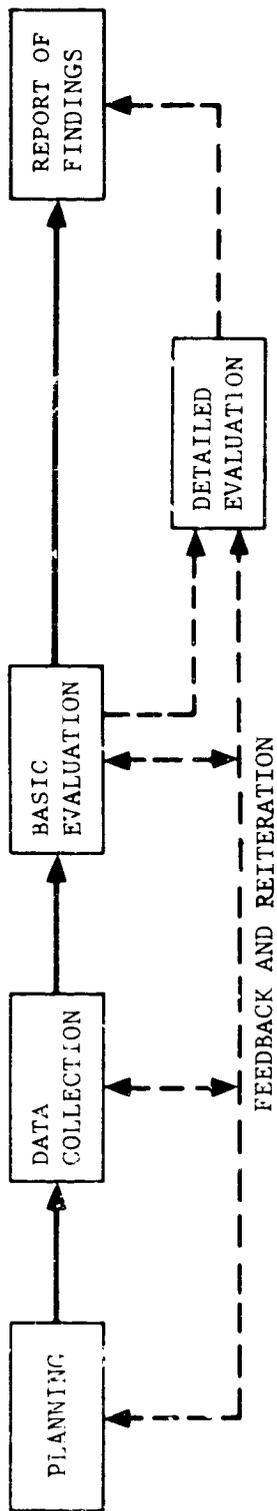
The test plan for security should describe what is to be tested, the testing schedule, resource requirements, testing materials, requirements for test training, the location of the test, the functional security requirements, the tests to be performed on the software and their relationship to the functional security requirements, the testing methodology, the evaluation criteria, data reduction techniques and a description of each test to be performed. The test plan format provided in FIPS PUB 38 [6], with slight modification, can be used to develop the security test plan. A suggested test plan format is provided in Appendix A.

At this point in the system development life cycle, only portions of the test plan can be developed. If the format at Appendix A is utilized, the following sections can be generated at this time. Section 1, General Information, can be completed. In Section 2, Plan, the software description can be written; tentative milestones can be developed; and, preliminary development of the testing subsection can be started. One of the critical items in Section 2 is the area of test training. Specifically, the types of training for the test team should be identified as soon as possible. In Section 3, under Specifications and Evaluation, the functional security requirements and the security functions to be tested can be identified. Additional portions of the test plan will not be able to be developed until such time as the

system/subsystem specifications, program specifications, data base specifications and security specifications have been generated.

One of the major objectives of the security test and evaluation of an application under development is to provide some of the data to support the certification of the security controls as required by OMB Circular A-71, Transmittal Memorandum No. 1. FIPS PUB 102, Guideline for Computer Security Certification and Accreditation [8], describes how to establish and carry out a certification and accreditation program for computer security. FIPS PUB 102 also provides some guidance that is useful in developing security test plans. It should be noted that in the NASA environment a different set of terminology is used in the area of certification and accreditation. In NASA, the term evaluation is used to identify the technical evaluation of the security of an application (synonymous with the FIPS PUB 102 term certification), and certification is used to identify the official management authorization for operation of the application (synonymous with the FIPS PUB 102 term accreditation).

FIPS PUB 102 defines the certification (evaluation) process as consisting of five activities: planning, data collection, basic evaluation, detailed evaluation, and report of findings. The process is summarized in Figure 2-6. The certification (evaluation) process is an iterative process. That is, based on findings from each stage, previous states might have to be reentered and work performed over. For example, basic evaluation might identify a function that is not included within evaluation boundaries but is important for security. This can require revision of the boundaries defined during



**FIGURE 2-6
CERTIFICATION PROCESS**

planning, along with additional data collection. The work is not sequential as suggested in the figure. Typically, most or all stages are ongoing at the same time. The intent of the figure is to show the shift in emphasis as work progresses.

It should be noted that basic evaluation or general evaluation is the minimum necessary for certification (evaluation) to take place. In general, basic evaluation suffices for most aspects of an application under review. However, most certifications (evaluation) also require detailed work in problem areas, and therefore require detailed evaluation as well. (For NASA, security tests are considered to be a detailed evaluation.) Minimum products required for certification and accreditation are a security evaluation report and an accreditation report.

2.3.1.7 Design the Security Specifications

The design stage is the time to make detailed decisions about how the security requirements will be implemented. There are usually a variety of ways to achieve an adequate level of security. In designing security controls, the age-old maxim of "Keep It Simple" is most applicable. A primary source of security problems is excessively complex design that cannot be implemented easily or correctly, and cannot be maintained nor audited. Lonsonsky [15] and Wong [16] suggest that, for any sensitive application, a thorough risk analysis, including safeguard selection, should be performed at the beginning of the design phase to assure that appropriate cost-effective controls are integral to the system's design. The guidance in FIPS PUB 65 [17] can be tailored for a risk analysis of an application system design [2]. FIPS PUB 73 [2] provides some ideas that apply to the overall security design effort.

- Unnecessary Programming. Terminals should be interfaced with the application system so as to minimize the danger that users can get unneeded programming capability. Users who can execute their own programs usually have the potential to bypass any security controls.
- Restricted User Interfaces. User interfaces should be tailored as specifically as possible to fulfill the user's requirements. Unneeded flexibility makes it more difficult to train users and to get them to accept the system. Flexibility also hurts security. The greatest danger occurs when users are given unnecessary access to a general purpose programming language; ...Once the user's needs are understood, interfaces should be designed to meet these needs as simply as possible with no unnecessary capabilities that complicate things both for the users and for the security analyst.
- Human Engineering. To preserve security and integrity, user interfaces must be designed so they are easy to understand and use. This can forestall many user errors, and it decreases the chances that users will neglect or bypass controls which they view as cumbersome and annoying.
- Shared Computer Facilities. It is easier to protect the code and data of the application system if it does not share computer facilities with other applications. It is especially useful to exclude all program development activities from the machine that runs the application--including the development and maintenance of the application's programs themselves.
- Isolation of Critical Code. The code and system data that is critical to security should be well identified so it can be more easily audited and protected. When possible, security controls should be isolated in modules that have few interactions with the rest of the application software. This makes it easier to audit these modules and protect them from unauthorized

modifications during operations. Sometimes automated controls can be used to protect these modules:

- * Checksums of the object code can be used to try to detect unauthorized changes.
- * Hardware protection states or protection domains can protect code and data critical to security.
- * If security-critical code always resides in a fixed area of memory, then read-only memory can be used.
- * Recent experimental languages, called type safe languages, may soon be available. Compilers for these languages can protect modules and their data from unexpected interactions with other modules.

Identification and partial isolation of critical code and data are reasonably easy; however, rigorous isolation is more difficult than it sounds. Without very careful planning, all system code and data will end up being relevant to security because errors or deliberate traps elsewhere can still cause security failures. This other software includes:

- * The operating system and other software that supports any of the security functions.
- * All parts of the application system needed to guarantee that the security controls are invoked at the appropriate time.
- * Compilers and other software used to develop and maintain any security-relevant software.

When code relevant to security is rigorously isolated from the bulk of the software, it is called a security kernel. Security kernels that protect data from unauthorized disclosure are feasible, but they require specially designed operating systems.

- Backup and Recovery. With appropriate contingency planning, the services of a computer application can usually be restored within a few hours after a failure. If availability requirements are more rigorous than that, then automatic backup and recovery mechanisms may need to be included in the application software.
- Use of Available Controls. The operating system and the facility management may already provide a variety of controls such as:
 - * User identity verification.
 - * Authorization for access to system files.
 - * Journaling of operating system activities.
 - * Backup and recovery operations.

While the application system usually needs to supplement these controls, available controls should be used to the fullest extent possible. In too many cases, controls are needlessly reimplemented because controls that are available are not understood or not utilized.

Since the security controls of an operating system are not absolutely reliable, the application system should use some data integrity checks to try to identify whether critical data has been altered; however, in general, there is no reason to believe that controls implemented in the application will be any more reliable than those already provided by an operating system.

The purpose of the design stage is to translate the security functional requirements into security specifications that can be used by programmers to develop the security-relevant code. Freeman [18] offers the following comments on the purpose of technical design.

...three basic purposes of design can be discerned...:

- discovery of problem structure;
- creation of the outlines (architecture, logical structure) of a solution for the problem;
- review of the results to ascertain if they meet the stated goals.

Once we understand the problem, the next major step is to develop outlines of the solution. This is the creative aspect of design in the strict sense of the word, although developing an accurate understanding of the problem requires just as much creativity in many cases.

The major activity is the establishment of the architecture of the system. That is, we engage in a combination of spelling out, in general terms, how the artifact will look to the user--the functions it will perform--and how it will be built--the major algorithms and data representations it will use.

Some parts of this process of spelling out the overall structure may require that we extend the design to a very detailed level in order to determine the feasibility of performing certain functions. But, in general, we are establishing the major pieces of the system, their relationships, interfaces to other systems and the outside world, and carefully specifying what must be done along with rough indications of how it is to be done.

The third purpose of design is to review repeatedly what has been done so far, to compare it to what is desired, and thus to evaluate progress. Review takes place at all stages of the development cycle, of course, but it is most central to the design phase. Review of code production is intended to determine that what has been implemented is what was specified; review of test results is meant to confirm that a sufficient set of tests has been run; review of specifications seeks to determine if the loosely stated requirements of the customer have been captured in operational terms. Review at the design phase, though, goes beyond just determining if something that has been previously spelled out has

been done--it is an integral part of the process of discovering the nature of the problem and the proper structure of the solution.

Freeman goes on to describe the software lifecycle as consisting of six stages: analysis, functional specification, architectural design, detailed design, implementation and evolution. Freeman's analysis of the development process, particularly the architectural design and detailed design are appropriate to understanding how to develop specifications for security requirements.

...For each stage, we will list the primary inputs (I), outputs (O), and major operations (OP)...

Architectural Design

- I: specifications, general context of desired system, knowledge of similar systems
- O: structural description of inside of system (definition of modules and interfaces)
- OP: discovery of problem structure, identification of major pieces of system, establishment of relationships between parts, abstraction, decomposition

Detailed Design

- I: architectural description, programming environment details
- O: blueprints for programs
- OP: abstraction, elaboration, choice of alternatives

In developing security specifications, there should be a normal evolution from architectural specifications to design specifications. Whether there will be an explicit distinction between architectural-type specifications and detailed specifications will depend on each organization's or Center's standards for software life cycle development. It should be noted that the terms logical design and physical design, used in some methodologies, correspond to SDLC architectural design and detailed design, respectively.

In Section 2.3.1.5, examples of security requirements were provided based on the basic controls described in FIPS PUB 73 [2]. The types of basic controls included: data validation, user identity verification, authorization, journaling, variance detection, and encryption. The following discussion will provide examples of specifications for some of the security requirements previously discussed.

- Requirement - All source data will be keyed twice and automatically compared with the transcribed source data previously keyed.
- Specification - A second person will key the data into a verifier. Only those fields containing transaction code, employee name, SSN, and grade will be verified. Any record containing a discrepancy between the initial keying and the verifying keying will be recorded on a discrepancy file. The discrepancy file will be forwarded to the input control group. Each record containing an error will be visually compared to the original source document and the input control group will resolve any discrepancy. The corrected transaction record will then be submitted input to the system.

(The operations manual would indicate the record positions of the fields to be verified and the

specific instructions for incorporating the correct record/transaction into the batching process.)

- Requirement - All keyed transactions (or transaction of a certain type) will be visually verified prior to transmission to the system.
- Specification - Data entry personnel will visually verify that the transaction code is equal to a 1, 2, or 3; the name field contains no arabic numeric data (note romanic number such as I, II, III, etc., are legal); the SSN field contains no alpha or special characters, no blanks; and that the grade field contains no alpha, special characters or blanks; that grade field contains only one of the following: 01, 03, 04, 05, 07, 09, 10, 11, 13, or 14.

If any field is not correct, the operator will check the source document for the correct data. If the source document is in error, the document will be returned to the point of origination for correction and resubmission.

- Requirement - All data transmitted between the host computer and remote site will be protected from unauthorized disclosure and modification during transmission.
- Specification - All data will be encrypted using the Data Encryption Standard as specified in FIPS PUB 46.

After the security specifications have been developed and approved, the development of test procedures can begin.

2.3.1.8 Develop Security Test Procedures

At this point in the life cycle, following the definition of the security specifications, Section 3 and 4 of the Security Test Plan can be developed. Using the test plan format at

Appendix A, the next step is to complete the sections dealing with methods and constraints and evaluation.

The methodology should indicate whether static or dynamic testing or both will be used, whether live or test data and an indication of the volume of data that is required to adequately test the security controls. The method for recording test results should be identified as well as any constraints that may limit the scope of the test. Under the evaluation section, the criteria for each type of test should be identified. Data reduction methods should also be described.

FIPS PUB 73 [2] offers some guidance on static and dynamic testing that is useful to this discussion of developing test procedures.

Static Evaluation. These techniques, which involve examination and analysis of the systems documentation and code, represent the most effective way to detect deliberate traps or other unauthorized modifications. However, due to the complexity of most systems and the limitations of automated techniques, and tools, it is not currently practical to analyze systems completely using static evaluation methods. In addition, static evaluation does not examine the system in a "live" or operational mode so that errors in the execution environment are not detected. Specific techniques and tools include:

- * Code Review. Portions of the source code are evaluated to determine if they implement the design specifications and are free from errors. In most cases, it will be impossible to review all of the

system's code. Generally, samples of code will be reviewed—especially critical modules or critical portions of the code. The code review can be done by an internal test and evaluation team that is involved in the system development or by an independent (third party) team—either internal or external to the organization. Code review differs from peer review, ...in that the code review is performed by individuals who were not involved with the design and programming of the application.

- * Penetration Studies. A few individuals can be challenged to find unknown weaknesses in the security controls. Penetration studies to identify programming errors can be expensive and are useful only if someone believes that there are no remaining errors that can affect security.
- * Source Code Analyzers. These software tools aid the evaluation process by providing details about specific characteristics of the source program.

Examples include:

- cross reference listings are an aid to code review and may be useful to identify "suspicious" variables and source statement references.
- the variables which can influence control flow decisions can all be identified or variables which could be read before any value has been assigned to them can be identified.

Dynamic Testing. These techniques involve executing the application system, or portions of the system, with test data and comparing the actual results with expected or known results. ...Dynamic testing is only practical for selected test cases. Fundamental questions such as, "Which test cases should be chosen?" and "How many test cases are enough?" must be answered. The answers to these questions depend upon the system being tested and upon the experience of the test team.

The following tools can be used to aid the dynamic testing process:

- * Program Analyzers. A program analyzer is a computer program that collects data about another program's operation while that program is executing. Program analyzers can be particularly useful for evaluating how thoroughly the test data has exercised the program being tested. In addition, they can be used to identify extraneous code that might be an unauthorized insertion.

- * Flaw Hypothesis Method. Security flaws can be hypothesized based on analogous flaws found in other systems and then tested for existence in this system. This is an effective approach for selecting test cases that are likely to find flaws.

The following excerpts for FIPS PUB 102 [8] on basic and detailed evaluation provide guidance that should be useful in developing test procedures.

...The general distinction between basic and detailed evaluation is that basic evaluation is primarily concerned with the overall functional security posture, not with the specific quality of individual controls. ...Basic evaluation is also concerned with verifying that security functions actually exist and that the implementation method is of sufficient quality to be relied upon. Detailed evaluation, on the other hand, is concerned with whether security functions work properly, satisfy performance criteria, and acceptably resist penetration.

There are four tasks in basic evaluation:

1. security requirements evaluation (are application security requirements acceptable?)
2. security function evaluation (do application security functions satisfy requirements?)
3. control existence determination (do the security functions exist?)
4. methodology review (does the implementation method provide assurance that security functions are acceptably implemented?)

Security Requirements Evaluation In both formulating and evaluating security requirements for an application, two classes of needs are considered: policy needs and situational needs. Policy needs derive from the principles and required practices that the application is obliged to pursue, such as Federal laws, regulations, standards and agency policies. Situational needs are those deriving from the application's characteristics and environment. To determine situational needs, four primary areas are considered: assets, threats, exposure and controls.

1. Asset. What should be protected?
2. Threats. What are assets being protected against?
3. Exposures. What might happen to assets if a threat is realized?
4. Controls. How effective are security safeguards in reducing exposures?

Security Function Evaluation. Given well-defined security requirements, function evaluation becomes the most important task in basic evaluation. It determines whether security functions (control techniques) such as authentication, authorization, monitoring, security management, and security labeling satisfy security requirements. The primary method is simply to use the stated requirements as a checklist to follow in assessing whether they are satisfied.

In some cases, requirements specify only the need for generic functions such as authentication. In other cases the requirements call for use of specific mechanism, such as particular password technique. In both situations, function evaluation identifies the defined security function and examines it for acceptability.

An important concern for function evaluation is the appropriate level of detail. The recommendation is that basic evaluations be complete (all applicable control features) down through the functional level, where "functional level" is the logical level represented by functions as defined in (or appropriate for definition) in the Functional Requirements Document. This notion applies to both controls within the computer and physical/administrative controls external to it (although the latter might not actually be defined in a Functional Requirements Document).

Control Existence Determination. The fact that functions are described in a document or discussed in an interview does not prove that they have been implemented. Basic evaluations require assurance that security function controls exist. The existence of most physical and administrative controls can be determined via visual inspection. For controls internal to the computer, testing is needed. . . . The intent is to simply verify that the functions exist.

Test for control existence determination are straight-forward. In many cases, a short operational demonstration suffices. For example, the existence of a password function can be determined by attempting to use the application and verifying that a valid password is required.

Methodology Review. Even though this is a high-level overview-type evaluation, it is still desirable to gain

some assurance that controls are acceptably implemented. The best way to do this without becoming immersed in testing or detailed analysis is to examine the methodology used to develop the application.

The areas of concern in reviewing a development methodology for certification are summarized below.

1. Documentation. Is there current, complete and acceptable-quality documentation?
2. Objectives. Was security explicitly stated and treated as an objective, with an appropriate amount of emphasis for the situation? Were security requirements defined?
3. Project Control. Was development well-controlled? Were independent review and testing performed and did they consider security? Was an effective change control program used?
4. Tools and Techniques. Were structure design techniques used (e.g., modularization, formal specifications)? Were established programming practices and standards used (e.g., high order languages, structured walk-throughs)?
5. Resources. How experienced in security were the people who developed the application? What were the sensitivity levels or clearances associated with their positions?

Detailed evaluations involve analysis of the quality of security safeguards. Primary tasks are examinations of the application from three points of view:

1. Functional Operation (Do controls function properly?)
2. Performance (Do controls satisfy performance criteria?)
3. Penetration Resistance (How readily can controls be broken or circumvented?)

Detailed evaluation consists of a collection of approaches. Selection of which to use depends primarily

on the threats and exposures of concern, rather than on the general characteristics or overall sensitivity of the application. To illustrate, if the primary concern is to protect secrets from an external penetrator, penetration resistance is stressed.

Functional Operation. Functional operation is the point of view most often emphasized in detailed evaluation since it assesses protection against human errors and casual attempts to misuse the application. Evaluations of function operation assess whether controls acceptably perform their required functions. Although testing is the primary technique in evaluating functional operation, other validation and verification techniques must also be used, particularly to provide adequate analysis and review in early phases of the application life cycle. Testing for functional operation examine areas such as the following:

1. Control operation (e.g., do controls work?)
2. Parameter checking (e.g., are invalid or improbable parameters detected and properly handled?)
3. Common error conditions (e.g., are invalid or out-of-sequence commands detected and properly handled?)
4. Control monitoring (e.g., are security events such as errors and file accesses properly recorded; are performance measurement of characteristics such as resource utilization and response time properly recorded?)
5. Control management (e.g., do security procedures for changing the security table work?)

To illustrate this testing, consider several of the tests needed to examine control operation of a password function:

1. Test whether access without a password is disallowed.
2. Test whether valid passwords are accepted and invalid passwords are rejected.
3. ...Test the interface between the password function and the access authorization function by testing

whether access is properly allowed or disallowed. For example, verify that valid passwords allow proper access and do not allow improper access, and that invalid passwords result in proper access restriction.

4. Test whether the system responds correctly to multiple invalid passwords.
5. Test whether system-initiated reauthentication functions correctly.

Functional operation includes the application's resistance to external errors. Therefore the test areas of primary interest include those interfaces across which errors might propagate:

1. man-man (e.g., operator messages)
2. man-system (e.g., commands, procedures)
3. system-system (e.g., intersystem dialogue)
4. process-system (e.g., calls)
5. process-process (e.g., interprocess calls)

Besides testing, there are other security evaluation tools and techniques that can be of use in examining functional operations. For example, software tools for program analysis, can be helpful in documentation analysis. Matrices can suggest ideas for test cases and scenarios. Checklists have utility in providing quick training as well as suggesting ideas for tests.

Formal verification is a technique that may be used during a detailed evaluation. Formal verification offers the hope of being able to mathematically "prove" that a functional design abides by a few simple security rules, and that lower levels of abstraction are consistent with the proven higher-level design performance. A number of qualitative factors are listed under the general heading of performance, which is the second area of concern in detailed evaluation. These are availability, survivability, accuracy, response time, and throughput. They can be applied to either individual controls or entire applications.

1. Availability. What proportion of time is the application available to perform critical or full services: Availability incorporates many aspects of reliability, redundancy, and maintainability. It is often more important than accuracy. It is especially relevant to applications with denial of service exposures as primary concerns (e.g., air traffic control, automatic funds disbursement, production control). Security controls usually require higher availability than other portions of an application.
2. Survivability. How well does the application or control withstand major failures or natural disasters? "Withstand" includes the support of emergency operations afterwards, and recovery actions to return to normal operation.
3. Accuracy. How accurate is the application or control? Accuracy encompasses the number, frequency and significance of errors. Controls for which accuracy measures are especially applicable are identity verification techniques and communication line handling techniques.
4. Response Time. Are response times acceptable? Slow control response time can entice users to bypass the controls. Examples of controls for which response time is critical are passwords (especially in distributed networks) and identity verification techniques.
5. Throughput. Does the application or control support required usage capacities? Capacity includes the peak and average loading of such things as users and service requests.

Penetration Resistance. The task here is to assess resistance against the breaking or circumventing of controls, where resistance is the extent to which the application and controls must block or delay attacks.

Assessment of penetration resistance can be the most technically complex of the detailed evaluation categories. It is best done to establish confidence in security safeguards. It can also be done to find and fix flaws. In both cases it:

1. provides an assessment of an application's penetration resistance;
2. helps to determine the difficulties involved in actually exploiting flaws; and
3. provides a clear demonstration of flaw exploitability (since it might not be clear from analysis whether, say, an asynchronous timing flaw can be exploited).

The objective of penetration-resistance evaluation is to identify externally exploitable flaws in internal security functions and the interfaces to them. Following are illustrative areas for this detailed examination:

1. complex interfaces
2. change control process
3. limits and prohibitions
4. error handling
5. side effects
6. dependencies
7. design modifications/extensions
8. control on security descriptors
9. execution chain of security services
10. access to residual information

Additional information on these areas can be found in the IBM Systems Journal paper entitled: "Penetrating an Operating System: A Study of VM/370 Integrity" [19].

The finalization of the test procedures will be a simultaneous activity by the test team while system developers are writing the security code and documenting the security safeguards.

2.3.1.9 Write Security Relevant Code

The precise way in which security safeguards will actually be implemented in the software code will depend to a great extent on the programming language used, installation standards, and personal programming style. Within the foregoing limitations there are some practices that should be followed to ensure that security relevant code is understandable, auditable, maintainable and testable.

The System Auditability and Control (SAC) Study [3] in discussing application system development controls provides some guidance that is useful to the development of security relevant code.

The adequacy and effectiveness of controls included in computer application systems are affected by the methods and procedures used during the system development process.

One of the areas discussed in some depth is the use of structured programming. The following is a summary of the SAC discussion of structured programming.

The objective in using structured programming techniques is to develop more usable and effective programs. "Usable" implies that the program can be read and understood by technical persons who did not write it, including users and EDP auditors. "Effective" implies that the program is designed to fit into an overall application system scheme so as to reduce redundancy and ensure processing efficiency.*

Structured programming is a technique for system builders that renders systems easier to build, maintain and alter. It is a discipline that is used primarily in the detail design and

programming states of the development process. As such, it uses a stepwise top-down approach, in which program modules are organized by functional specifications into a balanced hierarchical structure with minimum side effects on each module.

Structured programming involves a team approach to detailed design, with team members being used to "walk-through" the design and coding of components. The effect is that the design and code can be viewed by other than the originator to detect faulty logic, hard-to-follow code and the extent to which the design meets prespecified objectives.

Stanford Research Institute, authors of the SAC study, found the following major techniques were used by many organizations:

- Program Structure. A semistrict program structure allowing GO TO statements in a downward direction within a section domain.
- Statement Formatting. Fixed column indentation for both processor division and data division sections; in addition, a maximum of one verb per line and specific columns for operators.
- Peer Reviews. Structured walk-throughs whereby at least two peers completely trace or walk through the code generated by another programmer.
- Team Organization. The establishment of an integrated team consisting of one project leader/analyst, two programmer analysts, one to three programmers and one programmer librarian.

* Security controls should meet the usable objective. To be effective, in the above context, security controls should not unnecessarily reduce processing efficiency and be implemented in a way that they are part of the overall application scheme, and do not appear or operate as an interruption to the normal flow of the program or application processes.

- Top-Down Design. This technique consists of designing program logic by specifying higher level functions first and determining the subfunctions required to implement these higher level functions.
- Segmentation. During detail design and programming, it is advantageous to keep programs and modules in the form of routines called segments, with each segment having but one entry and exit.*
- Structured Coding. This approach or discipline is used to depict the process of coding whereby there are conventions used for syntax, program format, restricted and controlled branching, and disciplines on logic.
- Walk-throughs. The walk-through consists of a planned review of all system specifications and coding by peers of the developers. Walk-throughs have been found to be instrumental in uncovering a majority of errors during the pre-installation and test phases of a system. It is usually worthwhile to allow others to review specifications and code before a joint meeting. The review meeting can then become mainly a question answering and resolution session.
- Programmer Librarian. The programmer librarian actually serves the purpose of documenting all source codes. This function is responsible for getting codes keyed, updating the source library, and other general program documentation.

FIPS PUB 73 [2] recommends the following practices to enhance the security of application systems:

- Program Libraries. The program library catalogs and controls access to all versions of program modules as they are being developed. These control functions can

* The segmentation technique is particularly applicable to the coding of security relevant code.

be carried out by either manual or automated means. The program library can provide the following types of security controls during the programming stage:

- * permit only authorized persons access to program modules,
- * record all accesses (especially modifications) to program modules,
- * associate control data, such as record and byte counts, with program modules to facilitate detection of changes, and
- * enable comparison of current versions of modules with previous versions to identify code that has been changed.

More rigorous controls are needed as the program modules near completion, especially once review and testing has begun.

- o Redundant Computation. Critical computations can be checked by redundant processing to verify correctness of the result. Examples of local redundancy checks include:
 - * recalculation of a critical result by an alternate method;
 - * checking a calculated result for reasonableness and consistency with other data items; and
 - * examining extra attributes in retrieved data to ensure that the data item found was the one that was searched for.
- Program Development Tools. The choice of the programming language and of other programming tools can enhance the reliability and correctness of the final products. Proper selection and utilization of such

tools will help prevent programming errors from entering the source code. Some specific program development tools include:

- * High Level Programming Languages. Programming languages are especially useful if they support structured control flow, extensive data definition facilities, strong type checking, restricted scopes for program variables, and well-defined module calling conventions. Compilers for such languages can do extensive checking to identify program errors.
- * Preprocessors. Many of the advantages of high-level programming languages can be accomplished through a preprocessor. A preprocessor can be used to:
 - eliminate some of the more restrictive conditions in an existing language (e.g., allow structured flow in FORTRAN programs), and
 - provide automated quality control by checking that program modules meet the coding standards of this project.
 - Other Tools. Program development tools such as those that reformat source code, produce cross-reference listings and aid debugging are useful to help programmers manage the complexities.

2.3.1.10 Document Security Safeguards

Program documentation is needed during any software development; it is especially necessary for security-relevant code. FIPS PUB 73 [2] defines security-relevant code as:

- code that implements security controls;
- code that performs critical processing (e.g., check disbursement, real-time controls); and
- code that has access to critical or sensitive data during its execution.

Brill [13] discusses the documentation of security controls as follows:

You plan your controls; verify them through reviews with users and management; verify the operability of the controls through tests where you can; and make whatever modifications are necessary to get the best overall workable set of controls. Then make sure--again by review and test--that your documentation properly reflects the controls you want and the way that you want them to work.

The SAC [3] study provides the following thoughts on documentation that are applicable to the documentation of security controls.

Documentation is the process of describing on paper what functions an application system performs, how it performs them, and how the functions are to be used. The objectives of good documentation are to provide application system designers, implementers, testers, users and EDP auditors with a clear means of understanding all aspects of the application system.

Documentation...is important because it helps ensure correct and efficient processing within both data processing and user areas; it increases the ease and accuracy of computer program maintenance, and it provides auditors with an independent basis for evaluating application control.

One approach to documentation is used by a large food manufacturer. This organization makes use of a documentation test to be administered throughout the phases of system development. The primary purpose is to ensure that appropriate documentation exists for the major phases of system development;...

In addition to verifying the existence of sufficient documentation, this technique also assesses various aspects of the documentation, including the following:

- Does the documentation give evidence that processing controls will be adequate?

- Have sufficient controls been built into the system to allow effective operation and maintenance?
- Can the documentation be used as a basis to prove that controls over operation and maintenance are adequate?

It is recommended that the documentation of security-relevant code be contained in an independent document. The sensitivity of security-relevant code is such that it should be well-protected and access to the documentation should be restricted.

2.3.1.11 Conduct Security Test and Evaluation

Ideally, the conduct of the actual security test and evaluation will be performed by an organization independent of the system developers and users; that is, an independent validation and verification (IV&V) team, quality assurance personnel or an audit group. If such groups do not formally exist within the organization, a team of security evaluators may be formed for the purpose of conducting the security test and evaluation. Team skills that will be required include application analysts, testers, programmers, penetration specialists, VV&T specialists and security.

The actual test will consist of some combination of the following: document reviews, interviews, dynamic tests and penetration resistance tests. Each of these have been previously discussed.

Regardless of what combination of tests is used, it is important that the test team keep a list of documents reviewed, document all interviews, and document and maintain the results of any tests run against the application.

2.3.1.12 Write Security Test and Evaluation Report

The purpose of the Security Test and Evaluation Report is to document the test and evaluation results and findings; present the demonstrated capabilities and deficiencies of the security controls; and provide a basis for preparing the proposed security certification report.

The Security Test and Evaluation Report should be prepared by, or under the direction of, the security team leader. It will then be transmitted to the application computer security official (CSO). FIPS PUB 103 [8], provides a sample outline that can be used to develop the Security Test and Evaluation Report (see Figure 2-7). As noted in Figure 2-8, the sample outline contains five sections. Section 1, Introduction and Summary, briefly describes the application and summarizes the evaluation findings and recommendations. Section 2, Background, provides contextual information for the Application CSO including the security standards and policies that were applied. Also, it should include a list of the general functional characteristics of the application that generally influence its certifiability (e.g., the absence or presence of user programming). The scope of the evaluation and the assumptions and constraints on the test should be identified.

Section 3, Major Findings, should summarize the controls in place and their role in protecting the data and/or the application. It should emphasize those controls that were found to be effective. Any vulnerabilities found during the test should also be documented. The report should identify those vulnerabilities which should be accepted and those which should be corrected.

1. INTRODUCTION AND SUMMARY
 2. BACKGROUND
 3. MAJOR FINDINGS
 - 3.1 General Control Posture
 - 3.2 Vulnerabilities
 4. RECOMMENDED CORRECTIVE ACTIONS
 5. EVALUATION PROCESS
- Attachment A: Proposed Certification Statement

FIGURE 2-7
SAMPLE OUTLINE FOR A SECURITY EVALUATION REPORT

Section 4, Recommended Corrective Actions, identifies what additional controls should be considered and the anticipated costs of such recommendations. Section 5, Evaluation Process, should summarize the security test and evaluation process to include the types of tests that were conducted.

2.3.1.13 Prepare the Proposed Certification Statement

The proposed Certification Statement should summarize the recommendations, the acceptability of applications' security safeguards, restrictions (e.g., applications must be run as a stand-alone system), and/or corrective actions that must be accomplished prior to allowing the application to commence running.

3. SOFTWARE QUALITY ASSURANCE AND SECURITY

G. H. Myers, in *Software Reliability, Principles and Practices* [4], begins a discussion of the definition of software reliability as follows:

The most significant problem facing the data processing business today is the software problem that is manifested in two major complaints: software is too expensive and software is unreliable. Most computer professionals recognize the former problem as largely a symptom of the latter....

It is interesting to note that the software reliability problem as it exists today was observed in the early days of computing:

Those who regularly code for fast electronic computers will have learned from bitter experience that a large fraction of the time spent in preparing calculations for the machine is taken up in removing blunders that have been made in drawing up the programs. With the aid of common sense and checking subroutines, the majority mistakes are quickly found and rectified. Some errors, however, are sufficiently obscure to escape detection for a surprisingly long time [20].

This observation was published by three British mathematicians in 1952. Although software errors were encountered before 1952, this seems to be the first recognition of the reliability problem, that is, a considerable amount of time is required for testing, and, even after this, some software errors will remain undetected.

The immediate problem encountered in dealing with software reliability is one of the definition: What is a software error? What is software reliability?

Myers provides the following definition:

A software error is present when the software does not do what the user reasonably expects it to do. A software failure is an occurrence of a software error.

Software reliability is the probability that the software will execute for a particular period of time without failure, weighted by the cost to the user of each failure encountered.

One of the areas of software where errors can be least tolerated is that of security safeguards. One of the techniques that is currently being employed to improve the reliability of software is software quality assurance. This section provides a discussion of software quality assurance, the software quality assurance life cycle, and how software quality assurance can be employed to reduce the potential for incorporating unreliable security safeguards in application systems.

3.1 The Cost of Software Errors

Sorkowitz [14] indicates that to better manage the development and maintenance of ADP systems, it is important to have an understanding of how software costs are distributed throughout the total software life cycle. Sorkowitz provides the following:

Life cycle costs are documented as follows:

a. Initiation Phase

Very little research has been done in this area.

b. Development Phase

A number of independent studies [21] divide the development costs as follows:

1. analysis and design--40%
2. coding and unit testing--20%
3. system test and integration--20%

The above figures have sometimes been described in the literature as the 40-20-20 rule.

Testing

Studies have shown that the various testing phases can account for up to 50% of the total resources spent for the development of a software system.

In discussing the detection of errors, Sorkowitz indicates that:

The phase in the life in which errors are detected is very important. This may sound obvious, but there are severe penalties if this simple point is not well understood. The cost of correcting an error increases with time.

Myers [4] shows two relationships concerning error correction versus schedule times (Figure 3-1). The first relationship shows that the cost of correcting an error increases rapidly during the latter parts of the development cycle. However, a second and less-known relationship also follows the same general curve. The probability of fixing a known error incorrectly also increases rapidly during the latter stages.

In the first relationship, Wolverton and Putnam [21] noted that a requirement error detected in the design stage is 2-1/2 times more costly to fix than if detected in the requirement stage. This same error detected in the Unit test stage is five times more costly and if found during integration testing is 36 times more costly. If we consider the very special case of a security safeguard that is embedded in the software and that safeguard contains an error, we can see that the ultimate potential cost can increase dramatically (Figure 3-2).

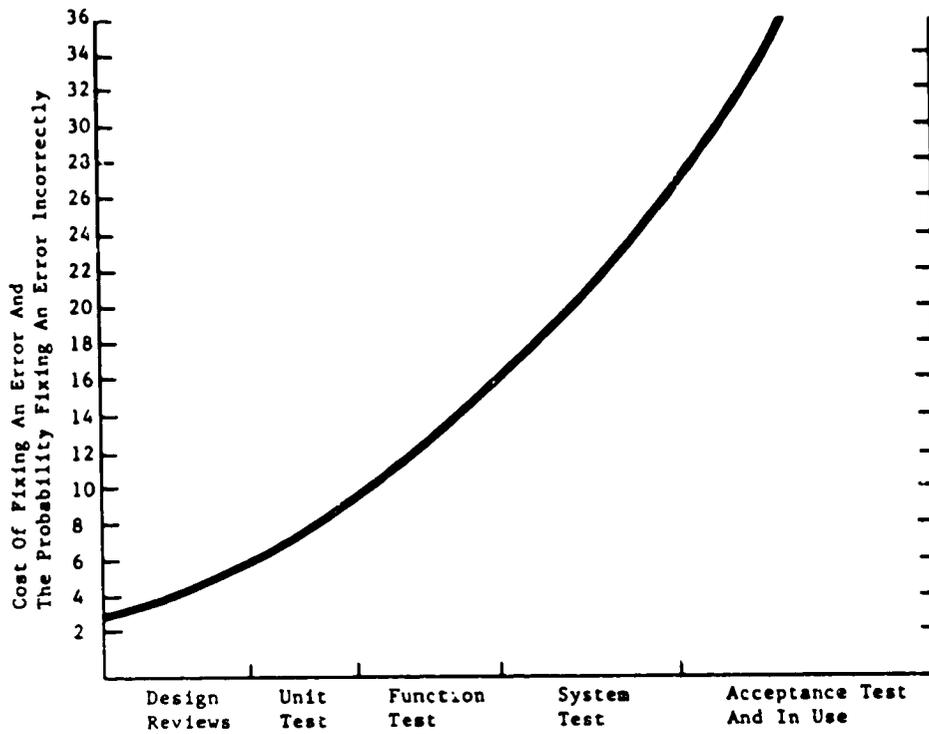


FIGURE 3-1
RELATIONSHIPS BETWEEN ERROR CORRECTIONS AND TIME

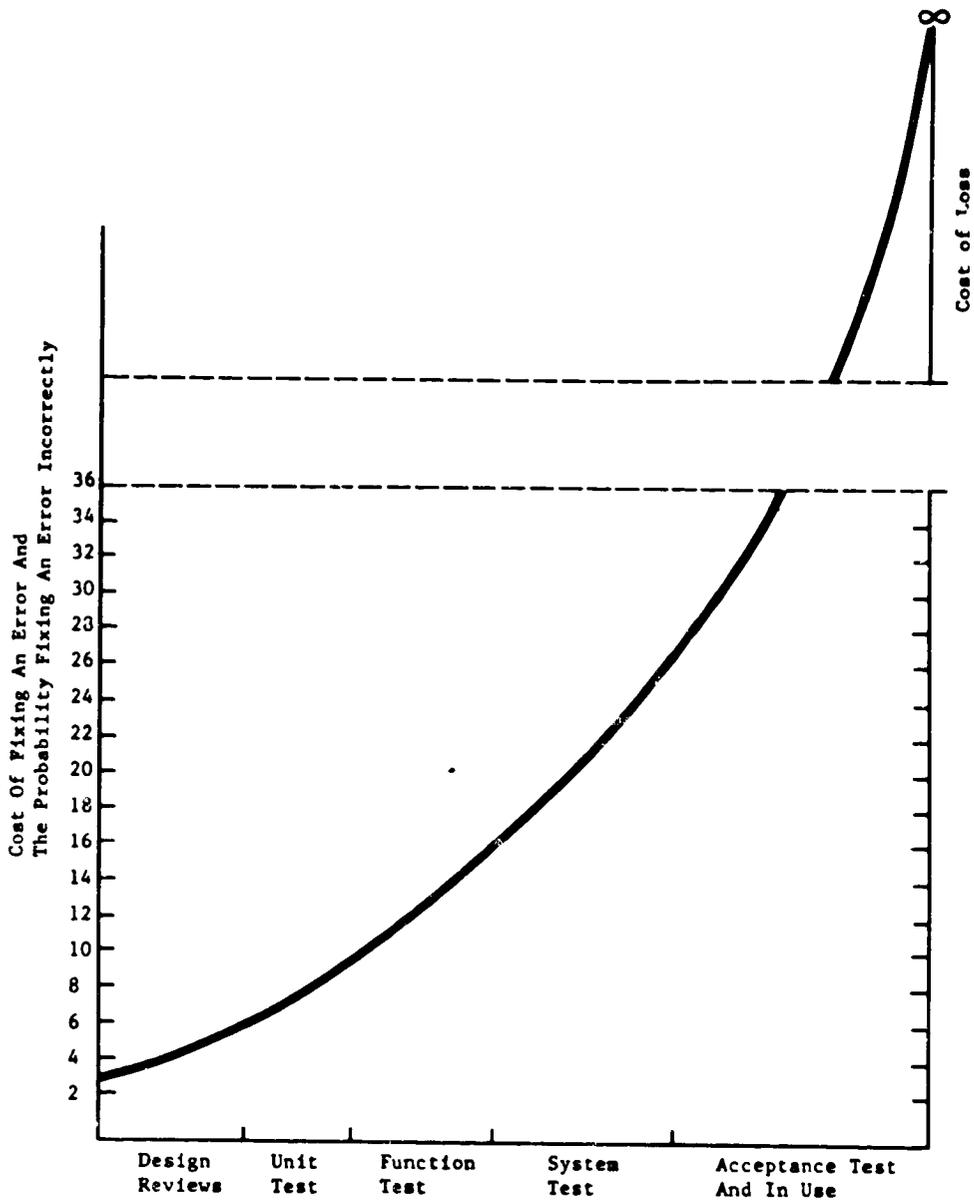


FIGURE 3-2
RELATIONSHIPS BETWEEN ERROR CORRECTIONS AND POTENTIAL
LOSS OF EXPLOITED ERRORED SOFTWARE SAFEGUARD

Not only will the organization incur the cost to fix the flawed software, but in addition, could incur a loss which may well exceed the cost to fix the flaw.

From the security perspective, the concern then is how do we ensure that the safeguards that are incorporated into application's software are free from errors and are reliable. From the system designer's and system developer's perspective, the concern is how do we define, design and develop error-free and reliable safeguards. In other words, how do we define and develop quality safeguards? One of the techniques is the use of a quality assurance process or function.

3.2 Software Quality Assurance

Perry, in Effective Methods of EDP Quality Assurance [22], provides some introductory comments on quality assurance that are appropriate to this discussion.

Organizations continually quest for quality products. Organizations that achieve a high level of quality in their products first establish an acceptable level of quality and then build a mechanism that assures this level is maintained. That mechanism in manufacturing is known as quality control. ...Quality control includes more than an evaluation of the end product. It begins with the examination of the raw materials and continues throughout the manufacturing cycle.

Data processing organizations must equate their function to manufacturing a product in order to see the need for a quality control function. Data processing must assume the responsibility of determining an acceptable level of quality, and then establish the mechanism (e.g., quality assurance function) to assure that level is maintained.

...the quality assurance function is an evolutionary step along the path of moving data processing from an art to a science.

One of the questions that comes to mind in any initial discussion of software quality assurance is: What is quality? Perry provides the following thoughts:

Quality is defined in the dictionary as an attribute or characteristic that is associated with something. Thus quality cannot be universally defined, but rather, must be defined for the item in question. Quality becomes a stated list of attributes and characteristics.

3.2.1 Software Quality Factors

To improve the quality of software, it is important to have an understanding of the attributes and characteristics, sometimes referred to as factors, that contribute to software quality. Sorkowitz [23] provides the following list of software quality factors.

- Correctness - the extent to which a program satisfies its specifications and fulfills the user's mission objectives.
- Reliability - the extent to which a program can be expected to perform its intended function with required precision.
- Efficiency - the amount of computing resources and code required by a program to perform a function.
- Integrity - the extent to which access to software or data by unauthorized persons can be controlled.
- Usability - the effort required to learn, operate, prepare input, and interpret output of a program.

- Maintainability - the effort required to locate and fix an error in an operational program.
- Testability - the effort required to test a program to ensure it performs its intended function.
- Flexibility - the effort required to modify an operational program.
- Portability - the effort required to transfer a program from one hardware configuration and/or software system environment to another.
- Reusability - the extent to which a program can be used in other applications--related to the packaging and scope of functions that programs perform.
- Interoperability - the effort required to couple one system with another.

3.2.2 Software Quality Factors and the Life Cycle

The points in the system development life cycle where each of the factors is of concern is illustrated in Figure 3-3. Figure 3-4 [23] identifies the relationship of software quality factors to the life cycle phases in terms of where quality factors should be measured and where the impact of poor quality is realized. It should be noted that software quality factors should be included in the functional requirements document and ultimately viewed as performance criteria.

3.3 The Software Quality Assurance Process

The purpose of employing a software quality assurance process is to improve computer software products that are produced via the software development process. Software quality assurance activities should be accomplished at pertinent points during

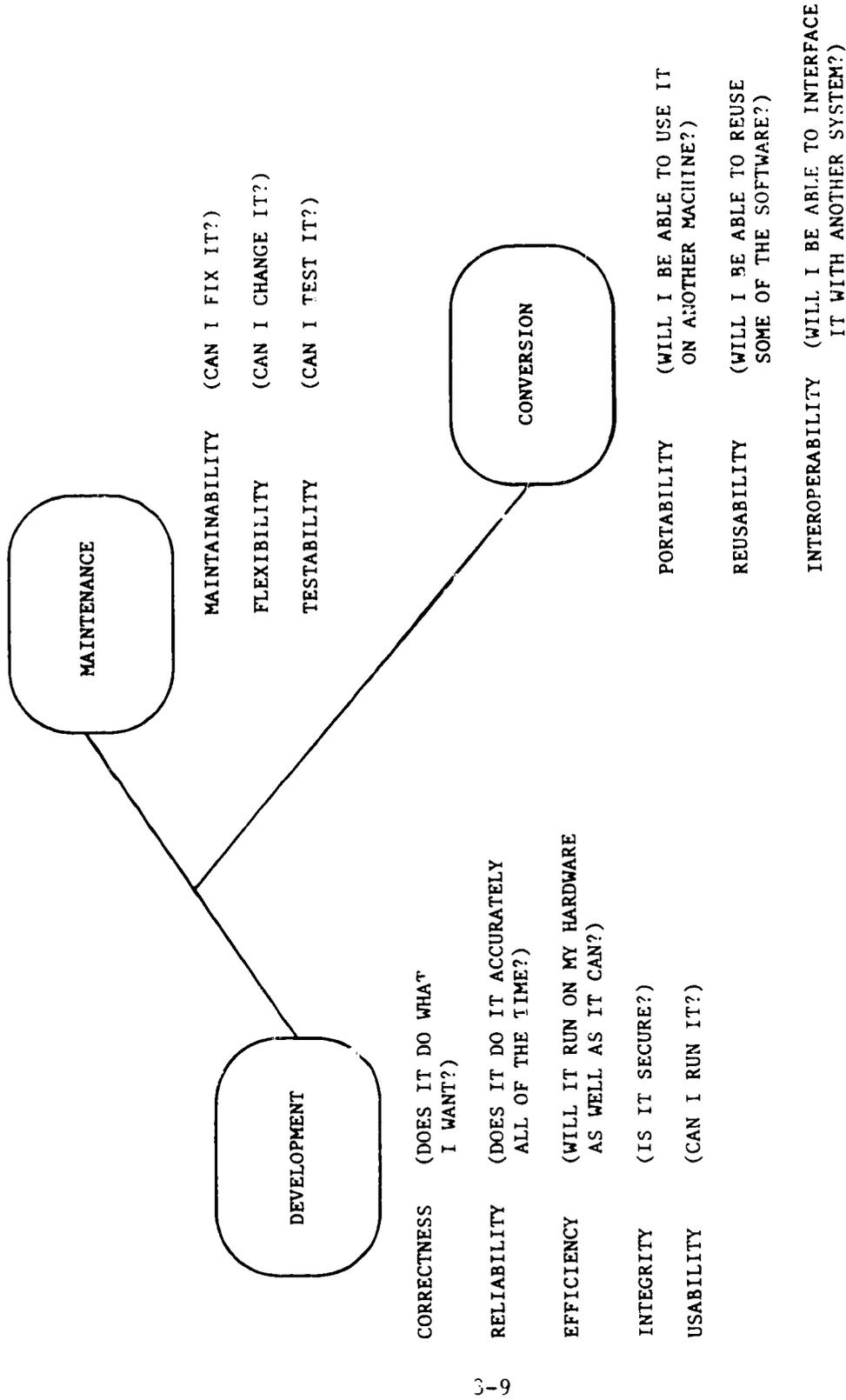


FIGURE 3-3
RELATIONSHIP OF SOFTWARE QUALITY FACTORS
TO THE SOFTWARE LIFECYCLE

LIFE-CYCLE PHASES FACTORS	DEVELOPMENT			EVALUATION	OPERATION		
	REQUIREMENTS ANALYSIS	DESIGN	CODE AND DEBUG		SYSTEM TEST	OPERATION	MAINTENANCE
Correctness	Δ	Δ	Δ	X	X	X	
Reliability	Δ	Δ	Δ	X	X	X	
Efficiency		Δ	Δ		X		
Integrity	Δ	Δ	Δ		X		
Usability	Δ	Δ		X	X	X	
Maintainability		Δ	Δ			X	X
Testability		Δ	Δ	X		X	X
Flexibility		Δ	Δ			X	X
Portability		Δ	Δ				X
Reusability		Δ	Δ				X
Interoperability		Δ			X		

Legend:

Δ Where quality factors should be measured

X Where impact of poor quality is realized

FIGURE 3-4
RELATIONSHIP OF FACTORS TO LIFECYCLE PHASES

the system development life cycle. Also, procedures should be established to control and track changes generated during the development cycle. Software quality assurance involves the use of various reviews and the establishment of baselines throughout the development cycle. The reviews and baselines are depicted in Figure 3-5.

3.3.1 Software Quality Assurance Baselines

The characteristics of an evolving system and its configuration items are defined and documented in increasing detail at logical transition points, or baselines, in the system development life cycle. At any time in the life cycle, all of the previously established baselines, together with approved changes to these baselines, constitutes the identification of the system and its configuration items. Five baselines are usually defined in the software assurance life cycle: functional, allocated, developmental, product and operational.

- Functional Baseline - marks the end of the initiation phase and the start of the definition stage of the development phase.
- Allocated Baseline - marks the end of the design stage and the start of the programming stage and is established by the detailed design specifications.
- Developmental Baseline - marks the end of the programming stage and the start of the test phase.
- Product Baseline - marks the end of the test stage and the start of the operations phase.
- Operational Baseline - marks the end of the implementation stage and the start of the maintenance phase and is established by the satisfactory demonstration of the application system in the operational environment.

INITIATION PHASE	DEVELOPMENT PHASE				OPERATIONS PHASE	
	DEFINITION STAGE	DESIGN STAGE	PROGRAMMING STAGE	TEST STAGE	IMPLEMENTATION STAGE	MAINTENANCE STAGE
<ul style="list-style-type: none"> • System Requirements Review 	<ul style="list-style-type: none"> • System Design Review 	<ul style="list-style-type: none"> • System Specification Review 	<ul style="list-style-type: none"> • Test Readiness Review <ul style="list-style-type: none"> - Unit - Module - Interface System 	<ul style="list-style-type: none"> • Functional Configuration Audit-1 • Physical Configuration Audit-1 • Formal Qualification Review-1 • Product Baseline 	<ul style="list-style-type: none"> • Functional Configuration Audit-2 • Physical Configuration Audit-2 • Formal Qualification Review-2 • Operational Baseline 	
<ul style="list-style-type: none"> • Functional Baseline 	<ul style="list-style-type: none"> • Allocated Baseline 	<ul style="list-style-type: none"> • Allocated Baseline 	<ul style="list-style-type: none"> • Developmental Baseline 			

FIGURE 3-5
SOFTWARE QUALITY ASSURANCE REVIEWS AND BASELINES

3.3.2 Reviews and Audits

The various reviews and audits that should take place throughout the system development life cycle include the system requirements review (SRR), system design review (SDR), preliminary design review (PDR), critical design review (CRD), test readiness review (TRR), functional configuration audit (FCA), physical configuration audit (PCA), and formal qualification review (FQR).

- Systems Requirements Review (SRR) - the objective of this review is to ascertain the adequacy of the system requirements. It should be conducted when a significant portion of the system functional requirements have been established.
- System Design Review (SDR) - this review should be conducted to evaluate the optimization, correlation, completeness and risks associated with the allocated technical requirements. Also included is a summary review of the system process which produced the allocated technical requirements of the planning for the next phase of the effort. This review should be conducted when the system definition effort has proceeded to the point where system characteristics are defined.
- Preliminary Design Review (PDR) - this review should be conducted for each system element to (1) evaluate the progress, technical adequacy, and risk resolution (on a technical, cost and schedule basis) of the selected design approach, (2) determine its compatibility with performance and requirements of the development specification, and (3) establish the existence and compatibility of the physical and functional interfaces among the other elements (personnel, equipment, facilities and computer programs).
- Critical Design Review (CDR) - this review should be conducted for each element when the detailed design is essentially complete. The purpose of this review is to (1) determine that the detailed design of the elements under review satisfies the performance requirements of the development specifications, (2) establish the

detailed design compatibility among the elements, (3) assess the producibility and risk areas (on technical, cost and schedule perspective), and (4) review the preliminary product specifications.

- Test Readiness Review (TRR) - a formal review should be conducted to validate the plan and the test procedures to include the test conditions, the extent of testing and the criteria for acceptance.
- Functional Configuration Audit (FCA) - a formal audit should be conducted to validate that the development of an element has been completed satisfactorily and that the element has achieved the performance and functional characteristics specified in the functional and design specifications.
- Physical Configuration Audit (PCA) - a technical examination of a designated element should be conducted to verify that the element "as built" conforms to the technical documentation which defines the element.
- Formal Qualification Review (FQR) - this review should consist of a test, inspection or analytical process by which products at the end item or critical end item level are verified to have met specific requirements (specifications or equivalent). This review does not apply to requirements verified during the FCA.

Reviews and audits should be conducted by personnel or an organizational entity independent of the development team.

3.4 Software Quality Assurance Life Cycle Security Activities

It is not sufficient to incorporate security safeguards in application systems. Safeguards should possess many of the same characteristics previously identified in the discussion of software quality factors. Also, security concerns should be integrated into the software quality assurance process in the same manner that security concerns should be incorporated into the system development life cycle process.

3.4.1 Security Safeguard Characteristics (Factors)

Security safeguards should possess the following characteristics: correctness, reliability, efficiency, integrity, usability, maintainability, testability, flexibility and interoperability.

- Correctness - the extent to which a security safeguard satisfies its specifications and fulfills the application security objectives.
- Reliability - the extent to which a security safeguard can be expected to perform its intended function with required precision.
- Efficiency - the amount of computing resources and code required by a security safeguard to perform its function.
- Integrity - the extent to which access to the security safeguard by unauthorized persons can be controlled.
- Usability - the effort required to learn, operate, prepare input and interpret output from a security safeguard.
- Maintainability - the effort required to locate and fix an error in or to determine the impact of other system changes on a security safeguard.
- Testability - the effort required to test or audit a security safeguard to ensure that it performs its intended function.
- Flexibility - the effort required to modify an operation security safeguard.
- Interoperability - the effort required to couple to or integrate security safeguards into the application system.

3.4.2 Security Assurance Activities

The security activities that should be an integral part of the software quality assurance process consist of a set of defined reviews and audits and approvals. The reviews and audits to be accomplished throughout the software quality assurance life cycle can be viewed as an integral part of the software quality assurance process but are separately identifiable actions. The security activities (Figure 3-6) to be completed in conjunction with software quality assurance activities are: security requirements review, security design review, security specification review, security test readiness review, and the security test and evaluation review.

3.4.2.1 Security Requirements Review

The objective of this review is to ascertain the adequacy of the security objectives, security feasibility and the preliminary security requirements. It should be conducted when a significant portion of the security requirements have been defined and in conjunction with the system requirements review (SRR).

The security requirements should be reviewed and approved by the application computer security official.

3.4.2.2 Security Design Review

This review should be conducted to evaluate the completeness and appropriateness of the technical security requirements. The review should also evaluate the technical risks of the safeguards that are being considered to meet the security

INITIATION PHASE	DEVELOPMENT PHASE				OPERATIONS PHASE	
	DEFINITION STAGE	DESIGN STAGE	PROGRAMMING STAGE	TEST STAGE	IMPLEMENTATION STAGE	MAINTENANCE STAGE
<ul style="list-style-type: none"> System Requirements Review 	<ul style="list-style-type: none"> System Design Review 	<ul style="list-style-type: none"> System Specification Review 	<ul style="list-style-type: none"> Test Readiness Review <ul style="list-style-type: none"> Unit Module Interface System 	<ul style="list-style-type: none"> Functional Configuration Audit-1 Physical Configuration Audit-1 Formal Qualification Review-1 	<ul style="list-style-type: none"> Functional Configuration Audit-2 Physical Configuration Audit-2 Formal Qualification Review-2 Operational Baseline 	
<ul style="list-style-type: none"> Functional Baseline 	<ul style="list-style-type: none"> Allocated Baseline 	<ul style="list-style-type: none"> Allocated Baseline 	<ul style="list-style-type: none"> Developmental Baseline 	<ul style="list-style-type: none"> Product Baseline 		
<ul style="list-style-type: none"> Security Requirements Review 	<ul style="list-style-type: none"> Security Design Review 	<ul style="list-style-type: none"> Security Specifications Review 	<ul style="list-style-type: none"> Security Test Readiness Review 	<ul style="list-style-type: none"> Security Test & Evaluation Review 		
<ul style="list-style-type: none"> Security Requirements Approval 	<ul style="list-style-type: none"> Security Design Approval 	<ul style="list-style-type: none"> Security Specifications Approval 	<ul style="list-style-type: none"> Security Test & Evaluation Plan Approval 	<ul style="list-style-type: none"> Security Safeguard Certification 		

**FIGURE 3-6
SECURITY ASSURANCE REVIEWS AND BASELINES**

requirements. The review should be conducted when the security definition effort has progressed to the point where the types of security controls that are proposed for the system have been identified and the initial or draft architectural security specifications have been developed. The system process which produced the architectural specifications should be reviewed. This review should be conducted in conjunction with the preliminary design review. The architectural specifications should be approved by the application computer security official.

3.4.2.3 Security Specifications Review

This review should be conducted for each security safeguard when the detailed security specifications are essentially complete. The purpose of the review is to (1) determine that the detailed design of the safeguards under review satisfy the performance requirements of the architectural specifications, (2) establish that the detailed design of the safeguards is compatible with the application system detailed design, (3) assess the producibility and risk areas (from a technical, cost and schedule perspective), and (4) review the preliminary security product specifications.

This review should be conducted in conjunction with the critical design review. The detailed design security specifications should be approved by the application computer security official.

3.4.2.4 Security Test Readiness Review

The purpose of this review is to validate the security test plan and test procedures to include the test conditions, the extent of the security test and the criteria for acceptance. The review should also determine the readiness of the security controls for testing to ensure that the security test and evaluation schedule can be met. The review should be conducted in conjunction with the test readiness review. The test plan and test procedures should be approved by the applications computer security official.

3.4.2.5 Security Test and Evaluation Review

The purpose of the security test and evaluation review is to evaluate the auditability of the records of the procedures, the accuracy of the data resulting from the tests, and the effectiveness of the tools and techniques used during the test. A useful set of criteria evaluating the security test and evaluation report is provided by FIPS PUB 102 [8] (Figure 3-7).

Resource Questions

1. How much of resources (e.g., time, money) were expended in the evaluation?
2. Who performed the evaluation? What are their qualifications? Might there be any reasons to question their objectivity?

Process Questions

1. What technical review mechanisms were used?
2. Have the findings and recommendations been properly coordinated?
3. What major tools and techniques were used? What other experiences have there been with them? Have resources been effectively allocated to tools, analysis, and presentation of findings?

Content Questions

1. Are the findings and recommendations reasonable?
2. What are other agencies doing in similar situations? Are Federal and agency requirements applicable to this application? Are there recent or proposed policy changes that are applicable? Do agency needs override user needs? What are the penalties for not complying with policies and requirements?
3. Did the evaluation focus on the those things of primary importance? What assurances are there that major problem areas have not been overlooked? Are there safeguards not considered by the evaluation activity that might influence the findings? Are the recommendations prioritized? What was the basis for prioritization?
4. Many residual vulnerabilities will exist. Have they been identified?
5. Are recommendations and judgments supported? Is the quality of supporting data shown?

FIGURE 3-7
CRITERIA FOR ASSESSING SECURITY EVALUATION REPORTS

4. SAFEGUARD VISIBILITY

Most application systems manage and control valuable assets (i.e., financial data, data about people, data on physical goods or other management or technical information). The need for comprehensive, cost-effective controls or safeguards is generally obvious. Applications traditionally get controls by chance, by user insistence, by auditor involvement or by system developer's recognition of the problem. Brill [13] has found:

As an auditor, I sometimes encounter systems that have pretty good controls even though not one of the systems developers ever thought about them. The controls somehow evolved.

Brill has also observed that some people assume that controls are different from everything else in the system, and that they stand out like a proverbial sore thumb. On the other hand, some people have claimed that controls aren't different; they are part of the solution to the user's problem and just mingle in with other systems requirements.

Brill's observations raise some interesting questions about how visible or identifiable security controls should be in the code (particularly the source code) and the documentation of an application system. This notion of visibility is driven by the needs of the application owner, system designers, system developers, system maintainers, operators, users, data providers, data custodians, auditors, and last, but by no means least, the need to protect the application and its data from the potential perpetrator. The requirements for visibility of safeguards is depicted in Figure 4-1.

VISIBILITY DRIVERS	DEVELOPMENT DOCUMENTATION			SOURCE CODE	POST-DEVELOPMENT DOCUMENTATION			OPERATIONAL APPLICATION SYSTEM
	SECURITY REQUIREMENTS	SECURITY SPECS	TEST PLAN & REPORT		USER'S MANUAL	OPERATIONS MANUAL	PROC MAINT MANUAL	
Application Owners	HIGH	HIGH			HIGH			HIGH
System Designers	HIGH	HIGH						
System Developers		HIGH	HIGH	HIGH				
System Maintainers		HIGH	HIGH	HIGH			HIGH	
Operators						LOW		
Users	HIGH							LOW
Data Providers								LOW
Data Custodians								HIGH
Auditors	HIGH	HIGH	HIGH	HIGH	HIGH	HIGH	HIGH	HIGH
Protection Against Perpetrators	VERY LOW	VERY LOW	VERY LOW	VERY LOW	VERY LOW	VERY LOW	VERY LOW	HIGH

FIGURE 4-1
SECURITY SAFEGUARD VISIBILITY REQUIREMENTS

4.1 The Needs of the Application Owner

For systems in development, the application owner has responsibility for identifying the sensitivity of the application and data, the security objectives, assessing the security risks, participating in the security feasibility analysis, and assisting in defining the security requirements. Application owners need to be able to see the articulation of the security concerns in a very visible sense. Once the system is developed, the owner needs to be assured that all other persons who use or have access to the system are properly controlled. Therefore, the application owner needs to be able to see that security concerns are sufficiently visible in the user manuals in order to discharge their responsibility for controlling access, modification, use and publication of specific data elements within an application. The owner must also have some way of assuring that they can properly discharge their responsibility relative to special handling and disposition of output products, and other administrative controls over the functional user. Functional users who interact and use the application in the discharge of their duties do not want security safeguards to be highly visible when the application is in operational use. The concern is that if security is too visible and users perceive that security will constrain them or unnecessarily or interfere with their use of the system, the users will attempt to find ways to bypass security or dilute the effectiveness of security (e.g., sharing of passwords).

4.2 The Needs of Systems Designers

System planners and designers have responsibility for developing the security requirements and specifications in

concert with the user based upon the security objectives. As in the case of the application owners, planners and designers need a clear and very visible articulation of the security safeguards in the documents they help to produce.

4.3 The Needs of Systems Developers

Systems developers (programmers) will need a clear understanding of the security requirements and specifications in order to develop the software source code. The security-relevant source code needs to be very identifiable for those who will be involved in the review (e.g., structured walkthroughs) of the code.

4.4 The Needs of System Maintainers

The personnel who will be responsible for maintaining the software over the operational life of the application will, in all likelihood, not be the personnel who were involved in the development of the software. The maintainers must have clear and understandable source code and documentation (requirements and specifications) to work with, so they can fully understand the nature of changes required in the security safeguards or the potential impact of other software changes that may affect the security safeguards.

4.5 The Needs of Computer Operators

The computer operators who will be involved in the actual operation of the application software need little or no understanding of how safeguards in the software actually work. Rather, they need only enough instruction in the operator's manual to be able to respond to and report security violations.

4.6 The Needs of Data Users

Data users may be required to participate in the identification of sensitive data, security requirements and back-up requirements. Their need for visibility will vary throughout the system development life cycle from high during the requirements definition phase to low in the operations phase. The articulation of the security controls in user documentation must be sufficiently identifiable and understandable so that user's management can be assured that their data is properly protected. However, the visibility of the security controls during the operation of the application should not be so visible that user's perceive that they are being unduly restricted or constrained in the ability to have access to the application and their data.

4.7 The Needs of Data Providers

Data providers are organizations that provide data to the application in order that the application can achieve its intended purpose. Data providers need only a basic knowledge about the requirements for security controls as it affects protecting data during the inputting operation of the application.

4.8 The Needs of Data Custodians

Data custodians are organizations or organizational elements that are responsible for maintaining the security and integrity of data and software while it is under the control of that organization. Data custodians must have a thorough understanding of the safeguards employed to protect data while it is in their custody.

4.9 The Needs of Auditors

Auditors are responsible for reviewing the adequacy of computer security programs, the adequacy of internal controls incorporated in applications systems and may be involved in the development of application software to ensure that the application is auditable once it is in operation. The auditors needs for visibility of safeguards is high throughout every activity connected with the life of an application.

4.10 The Needs for Protection Against Potential Perpetrators

Since the objective of potential perpetrators is to affect personal gain in the form of money or information, information about how safeguards are designed and implemented in the code should be protected from the potential perpetrator. The one area where safeguards should be highly visible to the perpetrator are in the actual operation of the application. Systems which have good security controls that are visible to the unauthorized user and the authorized user who might attempt to use the system in an unauthorized manner tend to discourage attempts to abuse or misuse the system.

5. SECURITY SAFEGUARDS IN PACKAGED SOFTWARE

NASA, like most organizations, does not rely solely upon internally or contractor-developed applications. The recent advances in technology have made micro and personal computers a viable solution for many of NASA's data processing needs. Along with the acquisition of micro and personal computers has come a number of software packages that have substantially decreased the time from identification of a problem until implementation of a computer-based solution. Unfortunately, not enough attention has been given to the problems inherent in applying a generalized application design to a specific organization's unique set of objectives and constraints. Bloom and Schneider [25] refer to part of the problem as the "package trap."

The "package trap" is the idea that a package itself solves the business problem. Its greatest danger is that once an application has been identified as amenable to a package-based solution, too little emphasis will be put on the analysis of the business problems that dictate the need for the system.

Bloom and Schneider [25] also observed:

Adaptations must be made to the systems development life cycle to facilitate a package-based solution.

McMenemy [26] points out another major concern with software packages:

Software packages are sold as a fast and easy alternative to in-house development. That impression is aided by claims of software vendors that their packages can be installed and running in three days. For your own protection, you must clarify the important difference between "installed" and "implemented."

"Installed" means that the software programs will reside on your computer, awaiting the information to make them functioning systems.

In other words, the programming and debugging are done. "Implemented" means that all information is loaded, all necessary interfaces have been programmed and tested, all systems and user personnel are competent in operating the system, all documentation is completed and the system has paralleled the old system, all documentation is completed and the system has paralleled the old system to prove the validity of the functions and information.

As you can see, the terms are very different in their definition, and more importantly, in their impact on the purchaser. Therefore, it should be better to think of a software package as a means of eliminating only the time spent on initial system design and programming, remembering that mass amounts of information must still be entered into the system.

As McMenemy [26] points out, acquiring packaged software eliminates only the systems design and programming stages of the classic systems development life cycle. The Initiation Phase, the Definition and Testing Stages of the Development Phase, and the Implementation Stage of the Operations Phase of the system development life cycle should still be completed.

5.1 System Development Life Cycle Activities for Packaged Software

Schick [27] indicates that the steps involved in selecting specific software packages vary with each situation, but a general procedure is applicable in almost all cases. Schick advised that the steps to be followed in selecting software packages include:

Classifying Needs. If the computer must accommodate several different applications, decide which application is most important and choose the software for it first. Then check to see what good software is available to handle the less important applications on the same computer. If two or more applications have equal priority, you may have to compromise by selecting software to handle both applications adequately.

It is also useful to decide how essential the requirements are. Requirements may be categorized as fixed, flexible, and optional. A fixed requirement means that certain functions must be performed in a specific way.

After defining and classifying requirements, write them down so that they can serve as guide in evaluating software packages.

Make a Requirements Chart. You can make a requirements chart by using the specifications already developed, paying particular attention to fixed requirements. The result of this step will be a uniform means of evaluating products.

Define Software Capacity. The amount of data you expect the software to handle needs to be added to the requirements chart. Software capacity limits are set by the software author to ensure that the program can run in the memory available on the computer configuration for which it was designed. The present and anticipated data capacity for the next year or two should be estimated, and these numbers added to the requirements chart.

Locate the Packages. Identify software products that meet, or come close to meeting, the requirements. Read product reviews in computer magazines, scan product advertisements in trade periodicals, visit local computer dealers, contact your industry or trade association, talk to associates who have bought computer software for requirements similar to yours or call your accounting firm. Another source of information is published directories of software.

Rank the Products. This based on a tabulation of how closely the products meet the requirements. To be sure that the software is right for your business:

- Purchase or borrow the manual, and read it to understand the capabilities and limitations of the software
- Attend a demonstration of the software, - if possible using some of your own data
- Consider whether the people who will use the software can follow the instructions provided or will require special training

One area of critical importance in considering a package^d software solution is the potential for some customizing. For example, how can a package be modified to accommodate a function outside the package's scope such as security? Bloom and Schneider [25] offer some advice on the subject of modification.

Even beyond the consideration of package modifications in the near term, it is important to look at the package as the basis of future system enhancements. The user must understand what functionality the system will support immediately and what functions the system may be modified to support in the future. The user must also recognize that the relative difficulty of accommodating new features will depend heavily on the package's technical architecture.

One significant opportunity presented by a package-based implementation is the ability to get some software up almost immediately. The vendor's "vanilla" software should be installed as soon as possible on the user's equipment. Once in place, this baseline software can be used to perform a wide array of functions, from modeling the production environment for hardware and communications analyses to prototyping user interfaces.

A corollary benefit of installing the package sooner is that it can be tested sooner. Do not assume that since a particular piece of software is in operation at several sites it will not contain bugs or undocumented features. A healthy dose of skepticism pays dividends. Develop a set of representative test data, and subject the "vanilla" package to thorough testing.

When designing modifications and enhancements to software packages, there are a few subtleties to keep in mind. To achieve the true cost/benefit of a package-based application, efforts should be made to minimize the amount of modification. This approach will frequently require the user to choose between restructuring existing procedures to work with the package and customizing the package. During the system design phase, provision should be made for the way in which future vendor releases will be incorporated into the system.

Extensive testing is just as necessary for package-based applications for custom systems. While the classic unit- or module-level test is not truly applicable to unchanged portions of the software package, it is appropriate for modules containing new or modified code. Consequently, a test plan should be developed during the system design phase which addresses the modular design of the new system as well as the business function it is meant to serve. It is crucial that the user participate heavily in all phases of testing.

Structured walk-throughs are also a necessity; even more so when they address modifications to the package. These design walk-throughs should include not only the project development team, but also the system's immediate users and those in the DP organization who will be charged with maintaining the new system.

As can be seen by the preceding advice of a number of experts, the purchase, installation and implementation of a software package should follow the classic systems development life cycle, with the exception of the Design and Programming Stages

(See Figure 5-1). When addressing the area of security, it is important that the life cycle activities not be overlooked or ignored.

5.2 Approaches for Addressing Security in Software Packages

Software packages present a special concern when it comes to security. Since packages attempt to present a somewhat generic solution for a large base of users, vendors have only a limited perspective of the sensitivity of the data that will be processed by their package once that package is installed and implemented in the user's organization. The organization considering the purchase of a packaged software product must initially assume that the product will not be able to provide an adequate and appropriate level of security. It is crucial that the potential buyer accomplish most of the security activities previously identified in Section 2.4 of these guidelines. In particular, the following activities should be accomplished:

- Determine the Sensitivity of the Data/Application
- Determine the Security Objective(s)
- Assess the Security Risks
- Conduct a Security Feasibility Study
- Define the Security Requirements
- Evaluate the Security Features of the Package
- Develop the Security Test Plan
- Develop the Security Test Procedures
- Document the Security Safeguards
- Conduct the Security Test and Evaluation
- Write the Security Test Analysis Report
- Prepare the Security Certification Report

INITIATION PHASE	DEVELOPMENT PHASE			OPERATION PHASE
	DEFINITION STAGE	PRODUCT TESTING STAGE	PRE-IMPLEMENTATION STAGE	
Classifying	Reqmts Chart Define Software Capacity Locate Packages Rank Products	Testing & Evaluation	Modification/ Customizing Pre-Implementation Testing	

**FIGURE 5-1
THE SOFTWARE LIFECYCLE FOR PACKAGED SOFTWARE**

The major difference in the activities identified above and the security activities discussed in Section 2.4 is the evaluation of security features of the package. To aid in the evaluation of the package's security features, it might be helpful to draft a set of architectural level specifications for security. These specifications could then be used as a "checklist" to determine if the package does provide the type of security safeguards required. If the package contains no security safeguards or only partially satisfies the security requirements and specifications, the potential buyer must look to other alternatives for providing the requisite level of protection. In large part, the alternatives will depend upon the type of security not provided by the application. For example, if the package does not provide for user identification and validation, a stand-alone security package designed especially for the micro or personal computer that will ultimately run the package should be considered, if such a security package is available.

5.3 Security Assurance and Certification of Packaged Software

The one activity that has the greatest potential for being overlooked when acquiring, installing and implementing a software package is that of quality assurance. The concern for software quality assurance, particularly with respect to the security activities, is based on the fact that those who use the application, in many cases, will be the ones who are actually designing, evaluating and operating the package. When systems are developed in-house from scratch, there is usually a very formalized development and software quality assurance process as described in Sections 2 and 3 of these guidelines. Users who acquire, install and implement software packages that

process sensitive data/applications are also required to have those systems certified by an application computer security official.

There is also concern for the integrity of the safeguards that are implemented in conjunction with the software package. It is, therefore, critical that some form of quality assurance be performed during the pre-acquisition, acquisition, testing and installation of a software package. Software package users should review Section 3 of these guidelines and develop a software quality assurance process at a level of detail appropriate for their package. Specifically, the following reviews and approvals should be accomplished:

- Security Requirements Review and Approval
- Security Design Review and Approval of the Inherent or Added Security Features
- Security Test Readiness Review
- Security Test and Evaluation Plan Approval
- Security Test and Evaluation Review
- Security Safeguard Certification

APPENDIX A
SAMPLE SECURITY TEST PLAN

A-1

NEXT PAGE BLANK

APPENDIX A

SAMPLE SECURITY TEST PLAN

1. GENERAL INFORMATION

- 1.1 **Summary.** Summarize the security functions of the software and the tests to be performed.
- 1.2 **Environment and Pretest Background.** Summarize the history of the project. Identify the user organization and computer center where the testing will be performed. Describe any prior security testing and note results that may affect this testing.
- 1.3 **References.** List applicable references, such as:
 - a. Project request (authorization).
 - b. Previously published documents on the project.
 - c. Documentation concerning related projects.
 - d. FIPS publications and other reference documents.

2. PLAN

- 2.1 **Software Description.** Provide a chart and briefly describe the inputs, outputs, and functions of the software being tested as a frame of reference for the test descriptions.
- 2.2 **Milestones.** List the locations, milestones events, and dates for the testing.
- 2.3 **Testing (Identify Location).** Identify the participating organizations and the location where the software will be tested.
 - 2.3.1 **Schedule.** Show the detailed schedule of dates and events for the testing at this location. Such events may include familiarization, training, data, as well as the volume and frequency of the input.
 - 2.3.2 **Requirements.** State the resource requirements, including:
 - a. **Equipment.** Show the expected period of use, types, and quantities of the equipment needed.

- b. Software. List other software that will be needed to support the testing that is not part of the software to be tested.
- c. Personnel. List the numbers and skill types of personnel that are expected to be available during the test from both the user and development groups. Include any special requirements such as multi-shift operation or key personnel.

2.3.3 Testing Materials. List the materials needed for the test, such as:

- a. Documentation.
- b. Software to be tested and its medium.
- c. Test inputs and sample outputs.
- d. Test control software and worksheets.

2.3.4 Test Training. Describe or reference the plan for providing training in the use of the software being tested. Specify the types of training, personnel to be trained, and the training staff.

2.4 Testing (Identify Location). Describe the plan for the second and subsequent locations where the software will be tested in a manner similar to paragraph 2.3.

3. SECURITY SPECIFICATIONS AND EVALUATIONS

3.1 Specifications

3.1.1 Requirements. List the security functional requirements established by earlier documentation.

3.1.2 Software Functions. List the detailed security functions to be exercised during the overall test.

3.1.3 Test/Function Relationships. List the tests to be performed on the software and relate them to the functions in paragraph 3.1.2.

3.1.4 Test Progression. Describe the manner in which progression is made from one test to another so that the entire test cycle is completed.

3.2 Methods and Constraints.

- 3.2.1 Methodology. Describe the general method or strategy of the testing.
- 3.2.2 Conditions. Specify the type of input to be used, such as live or test data, as well as the volume and frequency of the input.
- 3.2.3 Extent. Indicate the extent of the testing, such as total or partial. Include any rationale for partial testing.
- 3.2.4 Data Recording. Discuss the method to be used for recording the test results and other information about the testing.
- 3.2.5 Constraints. Indicate anticipated limitations on the test due to test conditions, such as interfaces, equipment, personnel, data bases.

3.3 Evaluation.

- 3.3.1 Criteria. Describe the rules to be used to evaluate test results, such as range of data values used, combinations of input types used, maximum number of allowable interrupts or halts.
- 3.3.2 Data Reduction. Describe the techniques to be used for manipulating the test data into a form suitable for evaluation, such as manual or automated methods, to allow comparison of the results that should be produced to those that are produced.

4. SECURITY TEST DESCRIPTIONS

- 4.1 Test (Identify). Describe the test to be performed.
 - 4.1.1 Control. Describe the test control, such as manual, semi-automatic, or automatic insertion of inputs, sequencing of operations, and recording of results.
 - 4.1.2 Inputs. Describe the input data and input commands used during the test.
 - 4.1.5 Outputs. Describe the output data expected as a result of the test and any intermediate messages that may be produced.

4.1.4 Procedures. Specify the step-by-step procedures to accomplish the test. Include test setup, initialization, steps, and termination.

4.2 Test (Identify). Describe the second and subsequent tests in a manner similar to that used in paragraph 4.1.

APPENDIX B
REFERENCES

B-1

NEXT PAGE BLANK

APPENDIX B

REFERENCES

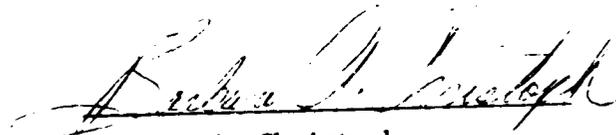
1. U.S. General Accounting Office. Report to Congress by the Comptroller General of the United States: Automated Systems Security - Federal Agencies Should Strengthen Safeguards Over Personal and Other Sensitive Data. Washington, D.C, U.S. General Accounting Office, 1979 January 23, LCD-78-123
2. National Bureau of Standards. Guidelines for Security of Computer Applications. Washington, D.C., GPO, 1980 June 30, FIPS PUB 73
3. Institute of Internal Auditors. System Auditability and Control Study - Control Practices. Altamonte Springs, FL, 1977
4. Glenford J. Myers. Software Reliability, Principles and Practices. John Wiley and Sons, New York, NY, 1976
5. Frank Mayo, Frederick G. Tompkins, Dana L. Hall. NASA Software Development and Assurance--Survey of Problems and Practices, MTR-82W205. The MITRE Corporation, McLean, VA, 1982
6. National Bureau of Standards. Guidelines for Documentation of Computer Programs and Automated Data Systems. Washington, D.C., GPO, 1976 February 15, FIPS PUB 38
7. National Bureau of Standards. Guidelines for Documentation of Computer Programs and Automated Data Systems for the Initiation Phase. Washington, D.C., GPO, August 1979
8. National Bureau of Standards. Guidelines for Computer Security Certification and Accreditation. Washington, D.C., GPO, September 1983
9. Alfred R. Sorkowitz. Software Life Cycle Costing. Proceedings Trends and Applications 1979. IEEE Computer Society, New York, NY, 1979
10. National Aeronautics and Space Administration. Computer Resources Management. NASA, Washington, D.C., NHB 2410.1
11. Brandt Allen. Threat Teams: A Technique for the Detection and Prevention of Fraud in Automated and Manual Systems. Computer Security Journal, Volume 1, Number 1, Spring 1981

12. Paul A. Giragosian, David W. Mastbrook, Frederick G. Tompkins. Guidelines for Certification of Existing Systems, MTW-82W18. The MITRE Corporation, McLean, VA, July 1982
13. Alan E. Brill. Building Controls into Structured Systems. YOURDON Press, New York, NY, 1983
14. Alfred G. Sorkowitz. Certification Testing: A Procedure to Improve the Quality of Software Testing. Computer, IEEE, August 1979
15. Terrance M. Losonsky. Automated Information System Security: A Comparative Analysis of Risk Management Procedures: Florida State University, December 1974
16. K. K. Wong. Computer Security Risk Analysis and Control, A Guide for the DP Manager, Manchester, England: The National Computing Center Limited; 1977. ISBN 0-8104-5466-1
17. National Bureau of Standards. Guideline for Automatic Data Processing Risk Analysis. Washington, D.C., GPO, August 1979. FIPS PUB 65
18. Peter Freeman. Introduction to: Tutorial on Software Design Techniques, Fourth Edition. Silver Spring, Maryland; IEEE Computer Society Press, 1983
19. C. R. Attanasio, P. W. Markstein, R. J. Phillips "Penetrating an Operating System: A Study of VM/370 Integrity." IBM Systems Journal, No. 1, 1976
20. R. A. Brooker, S. Gill, and D. J. Wheeler, "The Adventures of a Blunder," Mathematical Tables and Other Aids to Computation, VOL. 6 No1 38, 112-113, 1952
21. R. W. Wolverton and L. Putnam, "Quantitative Management: Software Cost Estimating," IEEE COMPSAC 77, Nov. 1977
22. William E. Perry. Effective Methodology of EDP Quality Assurance. Wellesley, M.A., QED Information Sciences, Inc., 1981
23. Alfred E. Sorkowitz. Life Cycle Quality Assurance Workshop, Potomac Forum, Alexandria, VA., 1984
24. Richard McCall. Factors in Software Quality, RADC-TR-77-369, VCL. I-II, Nov. 1977
25. Naomi Lee Bloom and Richard Schneider. Avoiding the 'package trap': Caution Advised, COMPUTERWORLD Special Report, January 30, 1984

26. K. J. McMenemy. Package Implementation Commands Planning, COMPUTERWORLD Special Report. January 30, 1984
27. Brian R. Schich. Tips on Shopping for Software, COMPUTERWORLD Special Report, January 30, 1984

Department Approval: 

This document has been peer reviewed by:


Barbara A. Christoph

DATE
FILMED

6 13 85

HB